Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

«Обработка разряженных матриц»

по курсу «Типы и структуры данных»

Студент: Шимшир Эмирджан Османович

Группа: ИУ7-33Б

Студент _____ Шимшир Э.О.

подпись, дата *фамилия, и.о.*

Преподаватель _____ Барышникова М.Ю.

подпись, дата *фамилия, и.о.*

Оценка _____

Условие задачи

Вариант 1

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов: - вектор ***A*** содержит значения ненулевых элементов;
- вектор ***JA*** содержит номера столбцов для элементов вектора ***A***;
- связный список ***IA***, в элементе Nk которого находится номер компонент

в ***A*** и ***JA***, с которых начинается описание строки Nk матрицы ***A***.

1. Смоделировать операцию сложения двух матриц, хранящихся в этой форме, с получением результата в той же форме.

2. Произвести операцию сложения, 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.

Техническое задание

Входные данные

Для корректной работы программы нужно ввести с клавиатуры или сгенерировать:

1. Матрицу №1

2.Матрицу №2

Каждая из этих матриц хранится как в обычном виде, так и в разреженном.

Выходные данные

Результирующие матрицы, полученные при сложении двумя способами выводятся на экран в обоих форматах.

Результаты измерения времени и затраченной памяти при выполнении сложения обоими методами также выводятся на экран

Задачи, реализуемые программой

1. Ввод матриц с клавиатуры
2. Генерация случайных матриц
3. Вывод матриц на экран в обоих форматах
4. Сложение матриц в обычном формате
5. Сложение матриц в разряженном формате
6. Вывод памяти и времи затраченных для обработки и хранения матриц двумя способами

Возможные аварийные ситуации и ошибки пользователя

Некорректный ввод пункта меню (вводить необходимо только целые числа).

Некорректный ввод целочисленных данных.

Попытка просмотра и сложения не проинициализированных прежде матриц.

Попытка сложения матриц разного размера

Описание внутренних структур данных

Структура для хранения матрицы в обычном виде:

```
typedef struct
{
    int rows;
    int columns;
    int non_zero;

    int **matrix;
} matrix_t;
```

int rows; - количество строк
int columns; - количество столбцов
int non_zero; - количество ненулевых элементов

*int **matrix;* - матрица элеменотв

В памяти матрица хранится в виде массива указателей на строки

Структура для хранения в разреженном виде:

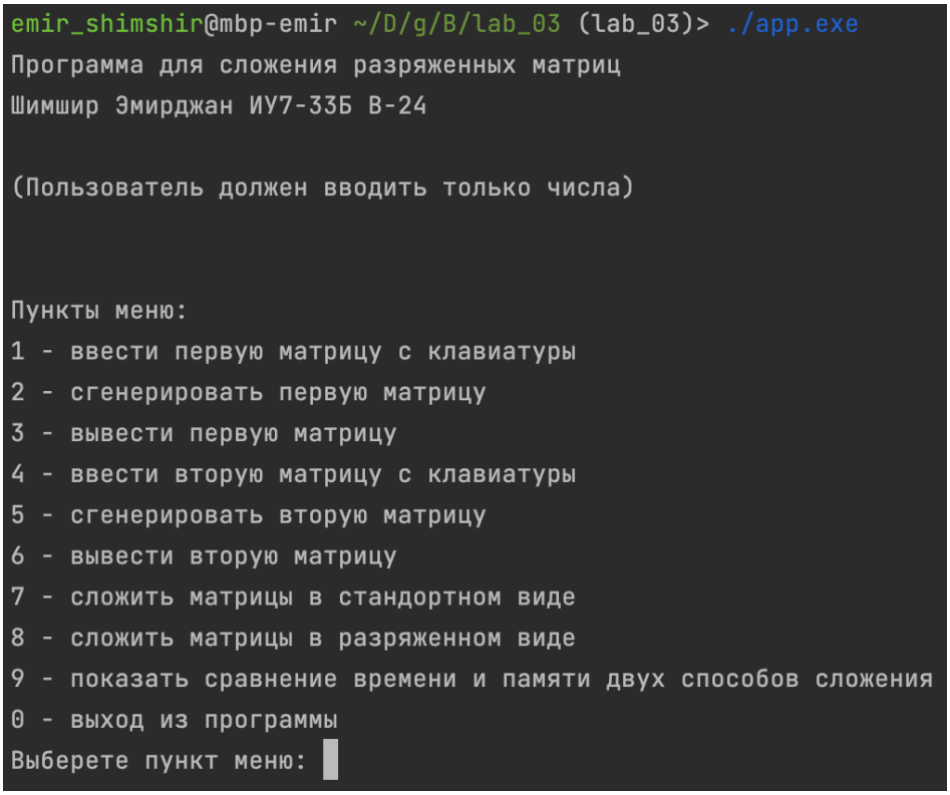
```
typedef struct
{
    int rows;
    int columns;
    int non_zero;

    int *A;
    int *IA;
    int *JA;
} sparse_matrix_t;
```

int rows; - количество строк
int columns; - количество столбцов
int non_zero; - количество ненулевых элементов

*int *A;* - массив ненулевых значений
*int *IA;* - массив с начальными индексами элементов массива A по строкам
*int *JA;* - массив индексов столбцов для элементов массива A

Память под описанные выше структуры данных выделяется динамически

Описание меню и функций программы



1. Пользователь выбирает пункт меню

2. Пользователь вводит матрицы одним из возможных способов (пункты 1, 2, 4, 5)

3. Для того, чтобы выполнить пункты 7-9 нужно проинициализировать обе матрицы

4. При выборе пунктов 3 и 6 можно увидеть соответствующие матрицы в двух форматах

5. Полученная матрица при сложении выводится на экран в двух форматах (пункты 7-8)

6. При пункте 9 выводится память и время затраченные для обработки и хранения матриц двумя способами

Описание кодов ошибок

Ниже описаны значения ошибок и их коды:

```c
#define EXIT_SUCCESS 0
#define ERR_MEM_ALLOC 1
#define ERR_ACTION 2
#define ERR_INPUT_ROWS 3
#define ERR_INPUT_COLUMNS 4
#define ERR_INPUT_NON_ZERO 5
#define ERR_INPUT_ELEM 6
#define ERR_INPUT_PRINT_METHOD 7
#define ERR_MATRIX_NOT_ALLOCATED 8
#define ERR_DIFF_MATRIX_SIZE 9
```

Ограничения на входные данные

Пользователю необходимо вводить только целые числа

Описание функций

```c
/*
* Функция заполняет матрицы с заданными параметрами случайными числами.
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
int generate_matrix(matrix_t *matrix, sparse_matrix_t *sparse_matrix);
```

```c
/*
* Функция выводит матрицу на экран
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
int print_matrix(matrix_t *matrix, sparse_matrix_t *sparse_matrix);
```

```c
/*
* Функция заполняет матрицы элементами, введенными пользователем
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
int read_matrix(matrix_t *matrix, sparse_matrix_t *sparse_matrix);
```

```
/*
* Функция заполняет матрицу в разряженной форме по данным из матрицы
* в стандартной форме
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
void fill_sparse_matrix_t_by_matrix_t(sparse_matrix_t *sparse_matrix,
matrix_t *matrix);
```

```
/*
* Функция заполняет матрицу в стандартной форме по данным из матрицы
* в разряженной форме
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
void fill_matrix_t_by_sparse_matrix_t(sparse_matrix_t *sparse_matrix,
matrix_t *matrix);
```

```
/*
* Функция складывает матрицы в стандартной форме
*
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
* Принимает указатель на структуру, хранящую матрицу в стандартном формате
*/
int addition_matrix_t(matrix_t *matrix_1, matrix_t *matrix_2);
```

```
/*
* Функция складывает матрицы в разряженном формате
*
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
* Принимает указатель на структуру, хранящую матрицу в разряженном формате
*/
int addition_sparse_matrix_t(sparse_matrix_t *sparse_matrix_1,
sparse_matrix_t *sparse_matrix_2);
```

Описание алгоритма и исследование полученных результатов

Описание алгоритма

Для сложения матриц в обычном формате используется стандартный метод сложения матриц. Происходит итерация по каждому элементу результирующей матрицы, в каждый из которых записывается результат сложения соответствующих элементов исходных матриц. Сложность данного алгоритма **O(n*m)** (n – количество строк, m – количество столбцов). Стоит заметить, что количество итераций в данном алгоритме никак не зависит от количества ненулевых элементов в исходных матрицах, и количество этих операций равняется **n*m**. Количество памяти необходимое для инициализации одной матрицы в обычном формате **n*m*sizeof(int) + n*sizeof(int*)**, при сложении в матриц в памяти необходимо выделить три таких матрицы.

Для сложения матриц в разряженном формате я использовал следующий алгоритм. Сначала я итерировал строки слагаемых матриц, а затем “сливал” данные в соответствующих строках в результирующую матрицу. При слиянии строк я опирался на упорядоченность индексов столбцов ненулевых элементов в рамках одной строки. При слиянии я использовал метод двух указателей, сложность которого **O(n)** (n – размер сливаемых строк). В худшем случае размер сливаемых строк равен размеру столбцов в матрице (при отсутствии нулевых элементов). Таким образом, сложность алгоритма **O(n*m)** (n – количество строк, m – количество столбцов). Стоит заметить, что количество итераций в данном алгоритме зависит от количества ненулевых элементов в исходных матрицах, и количество этих итераций равняется **non_zero** (non_zero – количество ненулевых элементов).

Полученные результаты

Результаты измерения времени и памяти при различном размере матриц и их заполненности. Для каждого случая проводилось 5 экспериментов. При проведении экспериментов программа была скомпилирована без оптимизаций (**-O0**), внешние задачи отсутствовали.

Время в микросекундах, память в байтах

Размерность		% заполненности		Разреженный формат		Обычный формат	
Матрица №1	Матрица №2	Матрица №1	Матрица №2	время	память	время	память
50x50	50x50	5	5	5	4540	10	31200
50x50	50x50	10	10	10	8428	11	31200
50x50	50x50	20	20	18	15860	11	31200
50x50	50x50	50	50	37	35572	11	31200
50x50	50x50	100	100	44	60612	11	31200
100x100	100x100	5	5	19	16988	42	122400
100x100	100x100	10	10	35	32420	42	122400
100x100	100x100	20	20	67	62004	43	122400
100x100	100x100	50	50	150	141004	42	122400
100x100	100x100	100	100	152	241212	43	122400
500x500	500x500	5	5	421	401084	1040	3012000
500x500	500x500	10	10	817	785876	1045	3012000
500x500	500x500	20	20	1613	1526468	1041	3012000
500x500	500x500	50	50	3652	3507020	1067	3012000
500x500	500x500	100	100	2930	6006012	930	3012000

Выводы из таблицы измерений:

Мы можем заметить, что алгоритм сложения в разреженном виде эффективнее по памяти и по времени при маленькой заполненности матриц (в моем случае при заполненности менее 10%). При 10% заполненности он примерно в 4 раза эффективнее по памяти и на 20% эффективнее по времени.

При матрицах, заполненных наполовину память при двух видах хранения практически равна. Что касается времени, время при сложении стандартном виде меньше в 3 раза, чем время при сложении в разряженном виде.

При полностью заполненных матрицах для разреженного вида нужно в 2 раза больше памяти, чем для обычного вида. Но при этом, разреженный вид не выигрывает в скорости, и в среднем медленнее в 3 раза.

.

Тестирование

Позитивные тесты

№	Входные данные	Действия и выходные данные	Результат
1	пункт = 0	Завершение программы	Код возврата - 0
2	пункт = 1 Ввод валидной матрицы вручную	Ввод количества строк, столбцов и ненулевых элементов матрицы. Ввод самих ненулевых элементов матрицы.	Ожидание следующего ключа
3	Ключ = 2 Генерация случайной матрицы	Ввод количества строк, столбцов и ненулевых элементов матрицы. Генерация случайной матрицы	Ожидание следующего ключа
4	Ключ = 3 выбор 0 матрица проинициализирована	На экран выводится матрица в стандартном виде.	Ожидание следующего ключа
5	Ключ = 3 выбор 1 матрица проинициализирована	На экран выводится матрица в разреженном виде.	Ожидание следующего ключа
6	Ключ = 4 Ввод валидной матрицы вручную	Ввод количества строк, столбцов и ненулевых элементов матрицы. Ввод самих ненулевых элементов матрицы.	Ожидание следующего ключа

7	Ключ = 5	Ввод количества строк, столбцов и ненулевых элементов матрицы. Генерация случайной матрицы	Ожидание следующего ключа
8	Ключ = 6 выбор 0 матрица проинициализирована	На экран выводится матрица в стандартном виде.	Ожидание следующего ключа
9	Ключ = 6 выбор 1 матрица проинициализирована	На экран выводится матрица в разреженном виде.	Ожидание следующего ключа
10	Ключ = 7 Обе матрицы проинициализированы	Сложение матриц в стандартном виде и вывод на экран результата в стандартном виде и разряженном виде	Ожидание следующего ключа
11	Ключ = 8 Обе матрицы проинициализированы	Сложение матриц в разряженном виде и вывод на экран результата в стандартном виде и разряженном виде	Ожидание следующего ключа
12	Ключ = 9 Обе матрицы проинициализированы	Вывод информации о памяти и времени для двух методов сложения матриц	Ожидание следующего ключа

Негативные тесты

№	Входные данные	Выходные данные	Результат
1	Ключ = 10 Неверный ввод ключа	Ошибка выбора пункта меню	Ожидание следующего ключа
2	Ключ = 7 Не обе матрицы проинициализированы	Ошибка, матрицы не проинициализированы	Ожидание следующего ключа
3	Ключ = 8 Не обе матрицы проинициализированы	Ошибка, матрицы не проинициализированы	Ожидание следующего ключа
4	Ключ = 8 Не обе матрицы проинициализированы	Ошибка, матрицы не проинициализированы	Ожидание следующего ключа
5	Ключ = 3 Матрица не проинициализирована	Ошибка, матрицы не проинициализированы	Ожидание следующего ключа
6	Ключ = 6 Матрица не проинициализирована	Ошибка, матрицы не проинициализированы	Ожидание следующего ключа

Контрольные вопросы

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

Разреженная матрица — это матрица, содержащая большое количество нулей.
Схемы хранения матрицы: сжатое хранение строкой (CSR - Compressed Sparse Row) (мой метод), Список координат (COO - Coordinate list), Словарь по ключам (DOK - Dictionary of Keys), Сжатое хранение столбцом (CSC - Compressed Sparse Column)

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

Под обычную матрицу (**n** – количество строк, **m** – количество столбцов) выделяет **n*****m***sizeof(int) + **n***sizeof(int*) ячеек памяти.
В случае разряженного формата требуется количество ячеек в размере **non_zero***sizeof(int) + n**on_zero** *sizeof(int) + (**n** + 1)*sizeof(int)

(**non_zero** - количество ненулевых элементов, **n** – количество строк).

3. Каков принцип обработки разреженной матрицы?

При обработке разреженной матрицы мы работаем только с ненулевыми элементами. Тогда количество операций будет пропорционально количеству ненулевых элементов (прямая зависимость).

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

Выбор наиболее эффективного способа хранения и обработки матриц зависит от её заполненности. Стандартные алгоритмы выгоднее применять при большом количестве ненулевых элементов (50% и больше).

Вывод

Были исследованы различные варианты хранения обработки матриц (в обычном и разряженном формате), описаны алгоритмы сложения матриц в обоих форматах

При маленькой заполненности матрицы выгоднее использовать хранение в разреженном виде. При средней заполненности матрицы виды хранения примерно равны по памяти, но по скорости выигрывает обычный вид хранения. При полной заполненности матрицы лучше всего использовать обычный вид хранения, он эффективнее по памяти и по скорости.