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 РАЗРЕЖЕННЫХ ДАННЫХ»

по курсу «Типы и структуры данных»

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент _____ Чепиго Д.С

подпись, дата *фамилия, и.о.*

Преподаватель _____ Барышникова М.Ю.

подпись, дата *фамилия, и.о.*

Оценка _____

Условие задачи

Вариант 6

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 компонентов 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.

Техническое задание

Входные данные

Для корректной работы программы нужно ввести или сгенерировать:

1. Вектор-строку

2.Матрицу

Каждый из этих двух параметров хранится как в обычном виде, так и в разреженной форме.

Вид хранения при обычном виде:

1. Вектор-строка

```
struct str_matrix
{
    int *mtr;
    int n;
    int n_zero;
};
```

int *mtr; - массив элементов вектора-строки

int n; - количество элементов вектора-строки
int n_zero; - количество ненулевых элементов вектора-строки

2. Матрица

```
struct matrix_full
{
    int **mtr;
    int n;
    int m;
    int n_zero;
};
```

int **mtr; - матрица элеменотв

int n; - количество строк
int m; - количество столбцов
int n_zero; - количество ненулевых элементов

Структура для хранения в разреженном виде:

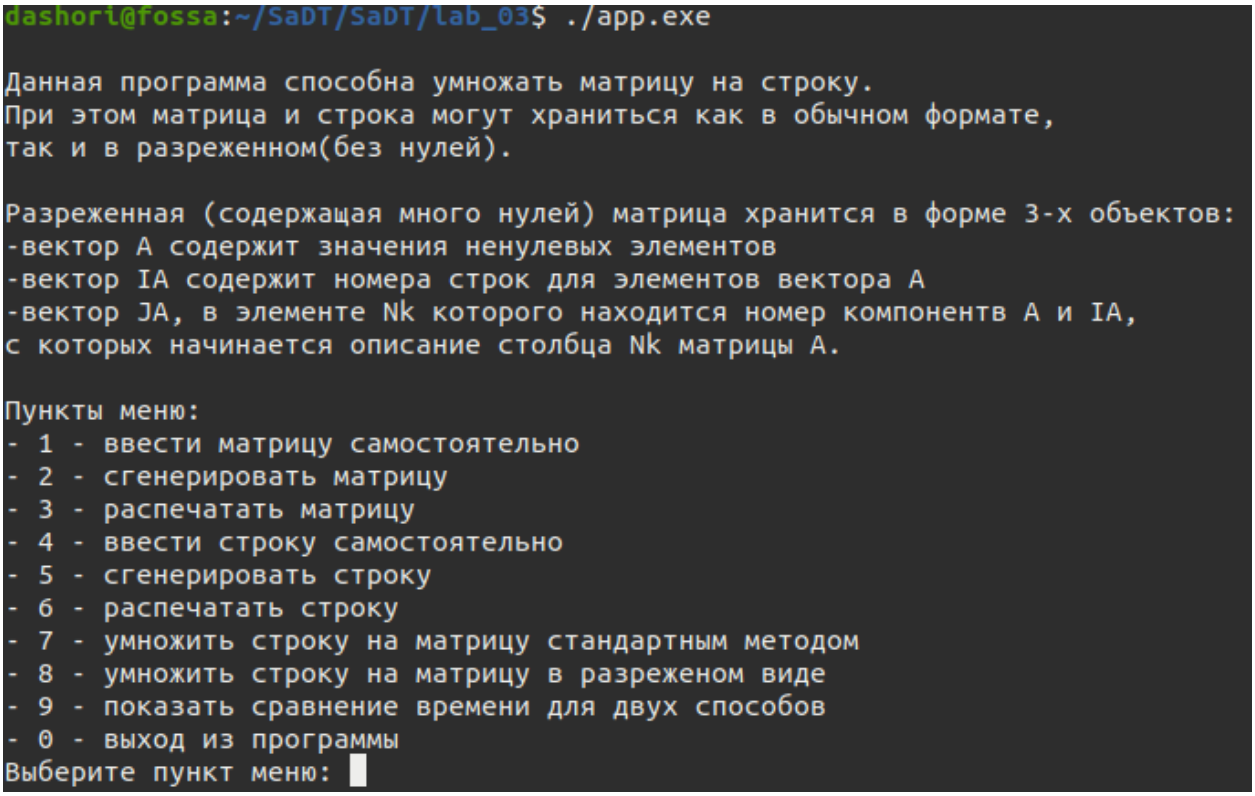
```
struct sparse_matrix
{
    int str;
    int col;
    int n_zero;

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

int str; - количество строк в матрице

int col; - количество столбцов в матрице
int n_zero; - количество ненулевых элементов

int *A; - массив ненулевых значений
int *IA; - массив индексов строк
int *JA; - массив с начальными индексами по столбцам

Описание меню и функций программы



Описание алгоритма

1. Пользователь выбирает пункт меню

2. Для того, чтобы выполнить пункты 7-9 нужно ввести матрицу и строку

3. Пользователь вводит матрицу и строку способом, который выбирает в меню

4. При выборе пунктов 3 и 6 можно увидеть текущую матрицу и матрицу-строку в двух форматах

4. Полученная матрица при умножении выводится на экран

5. При пункте 9 выводится память и время для обработки и хранения матрицы и строки

Сравнение эффективности

Время — в тактах

Память - в байтах

Размерность		% заполненности		Разреженный формат		Обычный формат	
Строка	Матрица	Строка	Матрица	время	память	время	память
5	5x5	15	15	5	100	8	160
5	5x5	50	50	5	172	8	160
5	5x5	100	100	6	284	8	160
10	10x20	15	15	7	380	11	920
10	10x20	50	50	10	964	9	920
10	10x20	100	100	15	1804	11	920
50	50x100	15	15	36	6668	73	20600
50	50x100	50	50	138	20804	84	20600
50	50x100	100	100	177	41004	97	20600
500	500x1000	15	15	2009	606604	8783	2006000
500	500x1000	50	50	13599	2008004	8714	2006000
500	500x1000	100	100	19825	4006404	8511	2006000

5000	5000x5000	15	15	25514	30046004	231603	100060000
5000	5000x5000	50	50	176790	104060004	228692	100060000
5000	5000x5000	100	100	223046	200080004	232144	100060000

Выводы из таблицы измерений:

Мы можем заметить, что алгоритм умножения в разреженном виде эффективнее по памяти и по времени при маленькой заполненности матрицы (в моем случае заполненность 15%). При 15% заполненности он примерно в 3 раза эффективнее по памяти и в 4 раза эффективнее по времени (при 5000x5000 элементах быстрее в 9 раз).

При матрице, заполненной наполовину память при двух видах хранения практически равна. Что касается времени, то на маленьких размерностях время примерно равное, но при размерностях больше 50x100 время при стандартном виде хранения быстрее на ~ 60%.

При полностью заполненной матрице 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 2 раза. Но, интересный момент, что на максимальной размерности и заполненности матрицы алгоритмы сравниваются по скорости, но не по памяти.

Вывод: при маленькой заполненности матрицы выгоднее использовать хранение в разреженном виде. При средней заполненности матрицы по памяти виды хранения примерно равны, а вот по скорости на небольших размерностях лучше использовать обычный вид хранения. При полной заполненности матрицы лучше всего использовать обычный вид хранения, он выгоднее и по памяти и по скорости.

Тестирование

Позитивные тесты.

№	Входные данные	Действия и выходные данные	Результат
1	Ключ = 0	Информация о завершении программы	Код возврата - 0
2	Ключ = 1 Ввод валидной матрицы вручную	Ввод количества строк, столбцов и ненулевых элементов матрицы. Ввод самих ненулевых элементов матрицы.	Ожидание следующего ключа
3	Ключ = 2 Генерирование рандомной матрицы	Ввод количества строк, столбцов и ненулевых элементов матрицы. Генерирование матрицы	Ожидание следующего ключа
4	Ключ = 3 выбор 1 матрицы существуют	На экран выводится матрица в стандартном виде.	Ожидание следующего ключа
5	Ключ = 3 выбор 2 матрицы существуют	На экран выводится матрица в разреженном виде.	Ожидание следующего ключа
6	Ключ = 4	Ввод количества столбцов, ненулевых столбцов и ввод самих элементов	Ожидание следующего ключа
7	Ключ = 5	Ввод количества	Ожидание

		столбцов, ненулевых столбцов и генерация самих элементов	следующего ключа
8	Ключ = 6 выбор 1 строка существует	Вывод строки в стандартом формате	Ожидание следующего ключа
9	Ключ = 6 выбор 2 строка существует	Вывод строки в разреженном формате	Ожидание следующего ключа
10	Ключ = 7 матрица и строка существуют	Умножение строки на матрицу и вывод на экран результата в стандартном виде	Ожидание следующего ключа
11	Ключ = 7 матрица существует, строка нет	Сообщение, что невозможно выполнить умножение без строки	Ожидание следующего ключа
12	Ключ = 8 матрица и строка существуют	Умножение строки на матрицу в разреженном виде и вывод на экран матрицы в разреженном виде	Ожидание следующего ключа
13	Ключ = 9 матрица существует, строка нет	Вывод информации об отсутствии строки	Ожидание следующего ключа
14	Ключ = 9	Вывод информации о затрачиваемой памяти и получившийся	Ожидание следующего ключа

		скорости для двух методов умножения	

Негативные тесты

№	Входные данные	Выходные данные	Результат
1	Ключ = 10 Неверный ввод ключа	Сообщение , что пользователь ввёл невалидный ключ.	Код возврата = 1
2	Ключ = 2 Ошибка выделения памяти	Завершение программы из-за невыделенной памяти	Код возврата = 2
3	Ключ = 1 Ввод букв или чисел вне границы (от 1 до 5000) вместо размера матрицы	Сообщение где именно пользователь ввёл невалидный результат.	Код возврата = 3
4	Ключ = 2 Ввод буквы вместо элемента матрицы	Сообщение где именно пользователь ввёл невалидный результат.	Код возврата = 5

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: связанная схема хранения (с помощью линейных связанных списков), кольцевая связанная схема хранения, диагональная схема хранения, строчной формат, столбцовый формат.

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) + M*sizeof(int*) ячеек памяти.
Для разреженной матрицы количество ячеек памяти завит от способа. В случае разряженного формата требуется количество ячеек в размере

K *sizeof(int) + K*sizeof(int) + (N + 1)*sizeof(int) (K — количество ненулевых элементов, N – количество столбцов).

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?

Эффективнее применять стандартные алгоритмы выгоднее при большом количестве ненулевых элементов(~50% от матрицы и больше).
Стоит отметить, что если расход памяти в программе не так важен, но важно время выполнения программы, то в случае умножения матрицы на вектор столбец лучше воспользоваться стандартным алгоритмом при большом количестве(~50% от матрицы и больше) ненулевых элементов в матрице, и умножение специального (разряженного) в случае небольшого количества(до ~50% от матрицы) ненулевых элементов.