Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
«Обработка разреженных матриц»

Студент, группа **Дегтярев А., ИУ7-33Б**

2020 г.

Описание условия задачи

Разработать программу умножения разреженных матриц. Предусмотреть возможность ввода данных, как с клавиатуры, так и использования заранее подготовленных данных. Матрицы хранятся и выводятся в форме трех объектов. Для небольших матриц можно дополнительно вывести матрицу в виде матрицы. Величина матриц - любая (допустим, 1000*1000). Сделать возможность заполнения разреженных матриц вручную (даже при большой размерности) и автоматически с разным процентом разреженности. Сравнить эффективность (по памяти и по времени выполнения) стандартных алгоритмов обработки матриц с алгоритмами обработки разреженных матриц при различной степени разреженности матриц и различной размерности матриц.

Техническое задание

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
- список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки 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.

Элементы матрицы вводятся координатным методом(номер столбца, номер строки, значение элемента), в любом порядке.

Входные данные

Для умножения матриц:

- Размер матрицы
- Элементы матрицы или процент заполнения матрицы для случайной расстановки ненулевых элементов

Для подсчёта времени выполнения алгоритма умножения:

- Размер матрицы
- процент заполнения матрицы

Выходные данные

Результат умножения для операции умножения матриц, время для подсчёта времени выполнения алгоритма умножения.

Функция программы

Программа позволяет умножать матрицу-строку на матрицу, используя разные виды представления матриц , и вычислять время выполнения программы.

Обращение к программе

В папке с исходным кодом в консоли нужно вызвать команду *make run*.

Аварийные ситуации

- Неправильный выбор команды
- Неправильный ввод матрицы
- Неподходящие для умножения матрицы
- Все элементы матрицы равны нулю.

Структуры данных

```
typedef struct matrix_s {
    int *data;
    size_t width;
    size_t height;
} matrix_t;
```

Тип *matrix_t* представляет способ хранения двумерных матриц в полном виде.

- *data* — массив элементов
- *width* — количество столбцов матрицы
- *height* — количество строк матрицы

Доступ к элементу по *y* индексу строки и *x* индексу столбца получается следующим образом: *data[x + y * width]*.

Эффективность структуры по памяти: *O(width * height)*.

```
typedef struct {
    int *values;
    size_t *columns;
    size_t *row_begins;
    size_t width;
    size_t height;
    size_t n_elems;
} smatrix_t;
```

Тип *smatrix_t* хранит матрицу в разреженном виде.

- *values* — массив элементов
- *columns* — массив содержит индексы столбцов для элементов *values*
- *row_begins* — массив содержит индексы элементов *values*, с которых начинается описание строк
- *width* — количество столбцов матрицы
- *height* — количество строк матрицы
- *n_elems* — количество ненулевых элементов

Доступ к элементу по *y* индексу строки и *x* индексу столбца получается следующим образом:

1. по *row_begins* определяется промежуток *[a, b)* массива *values*, в котором находятся элементы строки *y: (a = row_begins[y]; b = row_begins[y + 1])*
2. в промежутке [a, b) ищется индекс *k*, такой что *columns[k] == x*
3. *values[k]* — искомый элемент матрицы

Эффективность структуры по памяти: *O(n_elems + height)*.

Алгоритм

В начале идёт цикл меню действий. После выбора действия происходит ввод необходимых для действия данных, затем идёт основное действие.

Умножение полных матриц происходит обычным алгоритмом умножения матриц. Эффективность по времени: *O(w * n);* эффективность по памяти — *O(n);* где *w* - количество столбцов в матрице-строке, *n* - количество столбцов во второй матрице.

При умножении разреженных матриц:

1. Выделяем полную память подрезультирующую матрицу-строку
2. Рассматриваем *i*-ый элемент матрицы-строки, который имеет *y* индекс столбца.
3. Умножаем *i*-ый элемент на элементы второй матрицы, расположенных в *y* строке. Произведение добавляем в элемент результирующей матрицы, столбец которого совпадает с столбцом второй матрицы.
4. Убираем все нулевые элементы результирующей матрицы

Эффективность по времени: *O(w' * n);* эффективность по памяти — *O(n);* где *w'* - количество ненулевых элементов матрице-строке, *n* - количество столбцов во второй матрице.

При измерении времени каждый раз заполняем матрицы случайными числами.

Тесты

Группа	Входные данные	Выходные данные
Меню	отсутствующий пункт меню	Неправильный ввод
Ввод матрицы	Не положительный размер матрицы	Неправильный ввод
	Буква в размере матрицы	Неправильный ввод
	Координаты вне матрицы	Неправильный ввод
	Буква в координатах	Неправильный ввод
	Координаты элемента с нулевым значением	Матрица создана
	Буква в элементе	Неправильный ввод
	Буква в процент заполненности матрицы	Неправильный ввод
	Отрицательный процент	Неправильный ввод
	Процент > 100	Неправильный ввод
	Процент заполненности матрицы, при котором все элементы остаются нулевыми	Количество элементов не должно быть 0

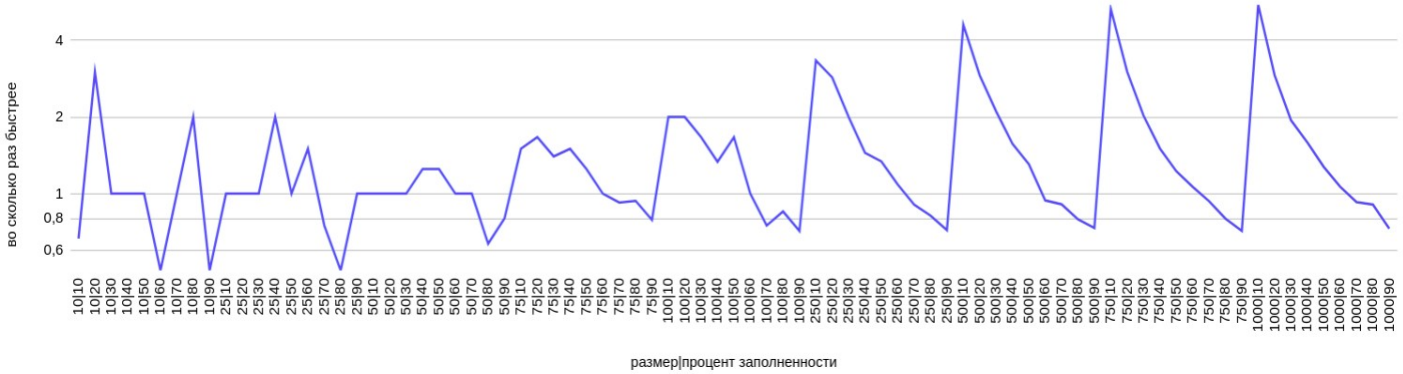
Оценка эффективности

Размер	Процент заполненности	Время умножения полных матриц, мкс	Время умножения разреженных матриц, мкс
10	10	4	6
10	20	6	2
10	30	1	1
10	40	1	1
10	50	1	1
10	60	1	2
10	70	1	1
10	80	2	1
10	90	1	2
25	10	1	1
25	20	1	1
25	30	1	1
25	40	2	1
25	50	2	2
25	60	3	2
25	70	3	4
25	80	2	4
25	90	3	3
50	10	2	2
50	20	3	3
50	30	3	3
50	40	5	4
50	50	5	4

50	60	6	6
50	70	7	7
50	80	7	11
50	90	8	10
75	10	3	2
75	20	5	3
75	30	7	5
75	40	9	6
75	50	10	8
75	60	11	11
75	70	12	13
75	80	15	16
75	90	15	19
100	10	4	2
100	20	8	4
100	30	10	6
100	40	12	9
100	50	20	12
100	60	18	18
100	70	21	28
100	80	23	27
100	90	25	35
250	10	20	6
250	20	40	14
250	30	52	26
250	40	68	47
250	50	87	65
250	60	101	93
250	70	115	127
250	80	132	161
250	90	146	203
500	10	69	15
500	20	134	46
500	30	200	95
500	40	259	165
500	50	334	256
500	60	385	409
500	70	448	493
500	80	527	664
500	90	582	793
750	10	154	29
750	20	295	98
750	30	435	215
750	40	584	388
750	50	730	596
750	60	869	817
750	70	1037	1110
750	80	1144	1433
750	90	1297	1816

1000	10	270	49
1000	20	522	179
1000	30	783	403
1000	40	1044	657
1000	50	1287	1010
1000	60	1547	1453
1000	70	1809	1951
1000	80	2289	2525
1000	90	2336	3206

Время умножения полных матриц к времени разреженных матриц



Процент заполненности	Память разреженной матрицы в процентах от полной матрицы (для больших размеров)
10	30
20	60
30	90
40	120
50	150
60	180
70	210
80	240
90	270

Функции

```
void print_menu(void)
      Выводит список действий в консоль

char *fgetline(char *dest, int maxlen, FILE *f)
      Считывает из потока непустую строку без символа окончания строки

void fwait_new_line(FILE *f)
      Считывает символы, пока не встретит символ окончания строки

int action_mul_matrix(void)
      Действие меню — умножение полных матриц

int action_mul_smatrix(void)
      Действие меню — умножение разреженных матриц
```

```
int action_measure_matrix_mul(void)
      Действие меню — измерение времени умножения полных матриц

int action_measure_smatrix_mul(void)
      Действие меню — измерение времени умножения разреженных матриц

matrix_t *create_matrix(size_t width, size_t height)
      Создаёт полную матрицу

int resize_matrix(matrix_t *m, size_t width, size_t height)
      Изменяет размер полной матрицы

void free_matrix(matrix_t *m)
      Освобождает выделенную под полную матрицу память

matrix_t *scan_matrix(void)
      Считывает полную матрицу из консоли

matrix_t *scan_row_matrix(void)
      Считывает полную матрицу-строку из консоли

int multiply_row_matrix_by_matrix(
    const matrix_t * restrict m_row,
    const matrix_t * restrict m,
    matrix_t * restrict result);
      Умножает полную матрицу строку на полную матрицу. Результат сохраняется в
result.

void print_matrix(const matrix_t *m)
      Выводит полную матрицу консоль

void fill_random_matrix(matrix_t *m, size_t n)
      Заполняет n элементов полной матрицы случайными элементами из промежутка
[-9, 9] \ {0}

smatrix_t *create_smatrix(size_t width, size_t height)
      Создаёт разреженную матрицу

int resize_smatrix(smatrix_t *m, size_t width, size_t height, size_t n_elems)
      Изменяет размер разреженной матрицы

void free_smatrix(smatrix_t *m)
      Освобождает выделенную под разреженную матрицу память

smatrix_t *scan_smatrix(void)
      Считывает разреженную матрицу из консоли

smatrix_t *scan_row_smatrix(void)
      Считывает разреженную матрицу-строку из консоли

int multiply_row_smatrix_by_smatrix(
    const smatrix_t * restrict m_row,
    const smatrix_t * restrict m,
    smatrix_t * restrict result);
      Умножает разреженную матрицу строку на разреженную матрицу. Результат со-
храняется в result.

void print_smatrix(const smatrix_t *m)
      Выводит разреженную матрицу консоль

void fill_random_smatrix(smatrix_t *m)
      Заполняет разреженную матрицу случайными элементами из промежутка [-9,
9] \ {0}
```

Вывод

Операция умножения разряженных матриц эффективнее по времени относительно умножения полных матриц для матриц >75x75 элементов и процентом заполненности <60% и эффективнее по памяти лишь при <30-40% заполненности. Таким образом, разреженные матрицы показывают свои временные преимущества при размерах > 75x75 и <30-40% заполненности.

Контрольные вопросы

1. Что такое разреженная матрица, какие способы хранения вы знаете?

Разреженная матрица – это матрица, содержащая большое количество нулей. Способы хранения: связная схема хранения, строчный формат, линейный связный список, кольцевой связный список, двунаправленные стеки и очереди.

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

Под обычную матрицу выделяет N * M ячеек памяти, где N – строки, а M – столбцы. Для разреженной матрицы – зависит от способа. В случае разреженного формата, требуется 3 * K ячеек памяти, где K – количество ненулевых элементов.

3. Каков принцип обработки разреженной матрицы?

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

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

Стандартные алгоритмы обработки матриц эффективнее применять при большом количестве ненулевых элементов (от 40%). Если не так важна память, занимаемая матрицами, но важно время, то для процентов заполненности, меньших 70%, лучше использовать алгоритм обработки разреженных матриц.