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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

“Обработка разреженных матриц”

Студент **Зернов Георгий Павлович**

Группа **ИУ7-34Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Зернов Г.П.**

Преподаватель _____ **Никульшина Т.А.**

2024

Оглавление

Условие задачи

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов (CSR): вектор A содержит значения ненулевых элементов; вектор JA содержит номера столбцов для элементов вектора A; вектор IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A. Смоделировать операцию сложения двух матриц, хранящихся в этой форме, с получением результата в той же форме. Произвести операцию сложения, применяя стандартный алгоритм работы с матрицами. Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц.

Техническое задание

Реализуемая в программе задача

Цель работы: реализация алгоритмов обработки разреженных матриц, сравнение эффективности применения этих алгоритмов со стандартными алгоритмами обработки матриц при различном размере матриц и степени их разреженности.

Входные данные

В случае выбора опции в меню целое число, соответствующее одной из указанных опций:

- 1 - Ввести (обновить) матрицу №1
- 2 - Вывести матрицу №1
- 3 - Ввести (обновить) матрицу №2
- 4 - Вывести матрицу №2
- 5 - Ввести (обновить) CSR матрицу №1
- 6 - Вывести CSR матрицу №1
- 7 - Ввести (обновить) CSR матрицу №2
- 8 - Вывести CSR матрицу №2
- 9 - Сложить матрицы
- 10 - Сложить CSR матрицы
- 11 - Анализ эффективности
- 12 - Выход

В случае ввода обычной матрицы:

- Число строк – целое число большее 0
- Число столбцов – целое число большее 0

- Элементы матрицы – последовательность целых чисел длиной равная произведению числа строк на число столбцов.

В случае ввода CSR матрицы:

- Число строк – целое число большее 0
- Число столбцов – целое число большее 0
- Число ненулевых элементов матрицы – целое неотрицательное число, не превышающее произведения числа строк на число столбцов
- Ненулевые элементы матрицы – последовательность троек целых чисел вида: <строка> <столбец> <значение> длиной равной числу ненулевых элементов матрицы. <строка> – целое неотрицательное число меньшее числа строк. <столбец> - целое неотрицательное число меньшее числа столбцов. <значение> - целое число не равное 0. В последовательности не должно быть элементов у которых равны соответствующие значения <строка> и <столбец>.

Выходные данные

1. Вывести матрицу №1 – выведенная на экран матрица №1.
2. Вывести матрицу №2 – выведенная на экран матрица №2.
3. Вывести CSR матрицу №1 – выведенная на экран CSR матрица №1.
4. Вывести CSR матрицу №2 – выведенная на экран CSR матрица №2.
5. Сложить матрицы – выведенная на экран матрица, равная сумме матриц №1 и №2.
6. Сложить CSR матрицы – выведенная на экран CSR матрица, равная сумме CSR матриц №1 и №2.
7. Анализ эффективности – выведенные на экран таблицы сравнения затраченных на сложение обычных и CSR матриц времени и памяти при разных значениях разреженности матриц.

Способ обращения к программе

Чтобы обратиться к программе, пользователю необходимо запустить файл app.exe.

Возможные аварийные ситуации или ошибки пользователя

- Невозможность интерпретировать ввод пользователя как валидное значение объекта, ввод которого был запрошен – вывод на экран информирующего сообщения о неверном вводе и предоставление пользователю возможности повторно ввести значение.
- Нехватка памяти при её выделении – соответствующее сообщение об ошибке и остановка текущего процесса с возвратом программы в состояние до его начала.

Внутренние структуры и типы данных

Структура, представляющее тип данных “вектор”.

- size_t elem_size – размер элемента вектора в байтах (целое число большее 0).
- size_t capacity – текущая вместимость вектора (в элементах) (целое число большее 0).
- size_t size – текущая длина вектора (в элементах) (целое число большее 0 и не превышающее capacity).
- void *data – динамический массив длиной capacity элементов по elem_size байт.

Листинг структуры:

```
typedef struct
{
    size_t elem_size;
    size_t capacity;
    size_t size;
    void *data;
} *vector_t;
```

Структура, представляющее тип данных “матрица”.

- size_t elem_size – размер элемента матрицы в байтах (целое число большее 0).
- size_t rows – число строк матрицы (целое число большее 0).
- size_t columns – число столбцов матрицы (целое число большее 0).
- void **data – динамический массив указателей на строки длиной rows. Строка – это динамический массив длиной columns элементов по elem_size байт.

Листинг структуры:

```
typedef struct
{
    size_t elem_size;
    size_t rows;
    size_t columns;
    void **data;
} *matrix_t;
```

Структура, представляющее тип данных “CSR матрица”.

- size_t columns – число столбцов матрицы (целое число большее 0).
- vector_t data – вектор ненулевых элементов матрицы.
- vector_t columns_data – вектор номеров столбцов ненулевых элементов матрицы. Номер столбца – целое неотрицательное число меньшее columns. Длины data и columns_data – равны.

• vector_t rows_ptrs – вектор указателей на строки, длиной на 1 больше числа строк в матрице. Указатель на строку – целое неотрицательное число, показывающее с какого элемента columns_data начинаются элементы этой строки.

Листинг структуры:

```
typedef struct
{
    size_t columns;
    vector_t data;
    vector_t columns_inds;
    vector_t rows_ptrs;
} *csr_matrix_t;
```

Алгоритм программы

- Общий алгоритм программы:

1. Запуск основного цикла программы.
2. Вывод меню.
3. Запрос у пользователя опции для выполнения.
4. Выполнение выбранной опции.
5. Переход к новой итерации цикла или завершение основного цикла при соответствующем выборе пользователя.

- Алгоритм ввода матрицы:

1. Запрос у пользователя количества строк.
2. Запрос у пользователя количества столбцов
3. Создание матрицы введённых размеров
4. Итерация по элементам матрицы и запрос у пользователя значения каждого элемента.

- Алгоритм ввода CSR матрицы:

1. Запрос у пользователя количества строк
2. Запрос у пользователя количества столбцов
3. Запрос у пользователя количества ненулевых элементов матрицы
4. Запрос у пользователя элементов матрицы с уникальными координатами для достижения требуемого числа ненулевых элементов.

- Алгоритм сложения матриц:

1. Проверка равенства размеров складываемых матриц
2. Создание матрицы результата
3. Итерация по элементам матрицы результата с записью в элемент суммы соответствующих элементов складываемых матриц.

- Алгоритм сложения CSR матриц:

1. Проверка равенства размеров складываемых матриц
2. Создание CSR матрицы результата
3. Копирование первой CSR матрицы в матрицу результата
4. Итерация по элементам второй матрицы, при наличии элемента с соответствующими координатами в матрице результата запись в него суммы элементов. Иначе вставка элемента второй матрицы на данные координаты в матрицу результата.

Функции программы

•int vector_create(vector_t *vector, size_t elemsize, size_t *init_capacity) – выделяет память под вектор элементов по elemsize байт, с начальной ёмкостью s(ize_t)*init_capacity, если передан NULL, то начальная ёмкость равна INIT_CAPACITY (10). Возвращает код ошибки.

•void vector_destroy(vector_t *vector) – освобождает выделенную под вектор память и присваивает ему значение NULL.

- size_t vector_size(vector_t vector) – возвращает текущую длину вектора.
- int vector_empty(vector_t vector) – возвращает является ли вектор пустым.
- void *vector_at(vector_t vector, size_t ind) – возвращает указатель на элемент вектора с указанным индексом.
- int vector_shrink_to_fit(vector_t *vector) – перевыделяет память под вектор так чтобы его ёмкость равнялась текущему размеру. Возвращает код ошибки.
- int vector_push_back(vector_t *vector, void *data, size_t data_size) – вставляет в конец вектора data_size элементов расположенных последовательно по указателю *data. Возвращает код ошибки.
- void vector_pop_back(vector_t *vector) – удаляет последний элемент вектора.
- int vector_insert(vector_t *vector, void *data, size_t ind) – вставляет на указанную позицию элемент. Возвращает код ошибки.
- int vector_copy(vector_t *dst, vector_t *src) – копирует вектор. Возвращает код ошибки.
- int matrix_create(matrix_t *matrix, size_t elemsize, size_t rows, size_t columns) – выделяет память под матрицу элементов размером elemsize байт с количеством строк rows и количеством столбцов columns.
- void matrix_destroy(matrix_t *matrix) – освобождает выделенную под матрицу память и присваивает ему значение NULL.
- size_t matrix_rows(matrix_t matrix) – возвращает количество строк в матрице.
- size_t matrix_columns(matrix_t matrix) – возвращает количество столбцов в матрице.
- void *matrix_at(matrix_t matrix, size_t i, size_t j) – возвращает указатель на элемент матрицы с координатами i и j.
- matrix_t matrix_sum(matrix_t *matrix1, matrix_t *matrix2, void (*sum)(const void *, const void *, const void *)) – возвращает сумму матриц, где операция суммы определяется функцией sum. В случае ошибки возвращает NULL.
- int matrix_fscan(FILE *file, matrix_t *matrix, size_t elem_size, int (*fscan_elem)(FILE * const, void * const)) – считывает из файла file матрицу элементов по elem_size байт, где fscan_elem – функция считывания элемента. Возвращает код ошибки.

- int matrix_user_fscan(matrix_t *matrix, size_t elem_size, int (*fscan_elem)(FILE * const, void * const)) – осуществляет пользовательский ввод матрицы элементов по elem_size байт, где fscan_elem – функция считывания элемента. Возвращает код ошибки.
- int matrix_fprint(FILE *file, matrix_t matrix, int (*fprint_elem)(FILE * const, void * const)) – выводит в файл file матрицу, где fprint_elem – функция вывода элемента. Возвращает код ошибки.
- int csr_matrix_create(csr_matrix_t *csr_matrix, size_t elemsize, size_t rows, size_t columns) – выделяет память под CSR матрицу элементов размером elemsize байт с количеством строк rows и количеством столбцов columns. Возвращает код ошибки.
- void csr_matrix_destroy(csr_matrix_t *csr_matrix) – освобождает выделенную под CSR матрицу память и присваивает ему значение NULL.
- size_t csr_matrix_rows(csr_matrix_t matrix) – возвращает число строк CSR матрицы.
- size_t csr_matrix_columns(csr_matrix_t matrix) – возвращает число столбцов CSR матрицы.
- void *csr_matrix_at(csr_matrix_t csr_matrix, size_t i, size_t j) – возвращает указатель на элемент CSR матрицы с координатами i и j. Если элемент нулевой, то возвращает NULL.
- int csr_matrix_put(csr_matrix_t *csr_matrix, void *data, size_t i, size_t j) – записывает в CSR матрицу элемент из указателя data по координатам i и j. Если по данным координатам есть элемент, он перезаписывается. Если координаты выходят за матрицу, то она расширяется. Возвращает код ошибки.
- csr_matrix_t csr_matrix_sum(csr_matrix_t *csr_matrix1, csr_matrix_t *csr_matrix2, void (*sum)(const void *, const void *, const void *)) – возвращает сумму CSR матриц, где операция суммы определяется функцией sum. В случае ошибки возвращает NULL.
- int csr_matrix_fscan(FILE *file, csr_matrix_t *csr_matrix, size_t elem_size, int (*fscan_elem)(FILE * const, void * const)) – считывает из файла file CSR матрицу элементов по elem_size байт, где fscan_elem – функция считывания элемента. Возвращает код ошибки.

- int csr_matrix_user_fscan(csr_matrix_t *csr_matrix, size_t elem_size, int (*fscan_elem)(FILE * const, void * const)) – осуществляет пользовательский ввод CSR матрицы элементов по elem_size байт, где fscan_elem – функция считывания элемента. Возвращает код ошибки.
- int csr_matrix_fprint(FILE *file, csr_matrix_t csr_matrix, int (*fprint_elem)(FILE * const, void * const)) – выводит в файл file CSR матрицу, где fprint_elem – функция вывода элемента. Возвращает код ошибки.
- int csr_matrix_copy(csr_matrix_t *dst, csr_matrix_t *src) – копирует CSR матрицы. Возвращает код ошибки.
- int time_measurement() – проводит анализ эффективности. Возвращает код ошибки.
- int matrixes_init(matrix_t *matrix, csr_matrix_t *csr_matrix, double density) – инициализирует матрицы по проценту ненулевых значений. Возвращает код ошибки.
- static void int_sum(const void *res, const void *num1, const void *num2) – возвращает сумму целых чисел расположенных по данным указателям.
- static int fscan_int(FILE * const file, void * const num) – считывает целое число из данного файла. Возвращает код ошибки.
- static int fprint_int(FILE * const file, void * const num) – выводит в данный файл выровненное целое число. Возвращает код ошибки.
- static int fprint_csr_int(FILE * const file, void * const num) - выводит в данный файл целое число. Возвращает код ошибки.
- void print_menu(void) – выводит меню.
- int main(void) – исполняет основной алгоритм программы.

Тесты

Первая матрица	Вторая матрица	Тестовый случай	Ожидаемый результат
		Введены невалидные данные на запрос программы	Сообщение о невалидности данных, пользователю предоставлена возможность ввести данные снова.
3	2 3 4 5	Матрицы разного размера	Сообщение о невозможности сложения матриц.

1 2 1 2 0 2	1 2 0 1 1	CSR матрицы разного размера	Сообщение о невозможности сложения матриц.
1 0 0 0 2 0 3 0 0 0 0 0 4 0 0 0 0 0 5 0 6 0 0 0 0	0 7 0 0 0 0 0 8 0 0 0 0 0 9 0 0 0 0 0 10 0 0 0 0 0	Сложение двух обычных матриц	1 7 0 0 2 0 3 8 0 0 0 0 4 9 0 0 0 0 5 10 6 0 0 0 0
1 2 3 0 2 1 0 2 3 3	4 5 6 1 0 2 0 1 3 3	Сложение двух CSR-матриц	1 4 5 2 6 0 1 0 2 2 0 2 3 5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Сложение двух обычных нулевых матриц	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - 0 0 0 0	- - 0 0 0 0	Сложение двух нулевых CSR-матриц	- - 0 0 0 0
1	4	Сложение двух матриц 1 на 1	5
7 0 0 1	6 0 0 1	Сложение двух CSR матриц 1 на 1	13 0 0 1

Анализ эффективности

Размер матрицы 10x10:

Процент заполнения	Время сложения обычных матриц (нс)	Время сложения разреженных матриц (нс)	Размер обычной матрицы (байт)	Размер разреженной матрицы (байт)
0	1897	1676	512	200
10	1864	1941	512	320
20	1881	2192	512	440
30	1877	2262	512	560
40	1871	2367	512	680
50	1848	2597	512	800
60	1850	2996	512	920
70	1892	3115	512	1040
80	1830	3380	512	1160
90	1899	3715	512	1280
100	1841	3834	512	1400

Размер матрицы 50x50:

Процент заполнения	Время сложения обычных матриц (нс)	Время сложения разреженных матриц (нс)	Размер обычной матрицы (байт)	Размер разреженной матрицы (байт)
10	11840	10734	10432	3520
20	11747	16817	10432	6520
30	11579	22866	10432	9520
40	12494	28020	10432	12520
50	11600	38727	10432	15520
60	12515	47080	10432	18520
70	11461	148939	10432	21520
80	11572	124537	10432	24520
90	11544	801344	10432	27520
100	11656	71377	10432	30520

Размер матрицы 100x100:

Процент заполнения	Время сложения обычных матриц (нс)	Время сложения разреженных матриц (нс)	Размер обычной матрицы (байт)	Размер разреженной матрицы (байт)
10	42890	31742	40832	12920
20	42016	584302	40832	24920
30	41052	1752675	40832	36920
40	39677	2925383	40832	48920
50	39676	3426599	40832	60920
60	40284	4253489	40832	72920
70	40061	3372748	40832	84920
80	39872	2418419	40832	96920
90	40075	1864790	40832	108920
100	40717	334323	40832	120920

Заметим, что при заполненности до 10% заполненности CSR матрицы эффективнее как по времени, так и по памяти. Затем до 30% заполненности

CSR матрицы эффективнее по памяти, но хуже по времени. При заполненности более 30% CSR матрицы во всём проигрывают обычным.

Вывод

При высокой разреженности разреженные матрицы эффективнее как по памяти, так и по времени. При уменьшении разреженности эффективность разреженных матрицы значительно падает и операции могут начать проводится с большими затратами времени и памяти чем для обычных матриц. При этом эффективность по времени падает быстрее, чем по памяти, в связи с чем обработка CSR матрицы может занимать несколько большее время, но быть эффективнее по объёму затраченной памяти.

Контрольные вопросы:

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

Разреженная матрица – матрица, большая часть элементов которой равна 0.

Схемы хранения соответствующих матриц:

- Словарь по ключам (DOK — Dictionary of Keys) – словарь, где ключ — это пара (строка, столбец), а значение — это соответствующий строке и столбцу элемент матрицы.
- Список списков (LIL — List of Lists) – список строк, где строка — это список узлов вида (столбец, значение).
- Список координат (COO — Coordinate list) – хранится список из элементов вида (строка, столбец, значение).
- Сжатое хранение строкой (CSR — Compressed Sparse Row) – матрица хранится в виде 3 списков: список значений, список координат по столбцам, список указателей на начала строк.
- Сжатое хранение столбцом (CSC — Compressed Sparse Column) – аналогично CSR, но строки и столбцы меняются местами.

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

Для обычной матрицы выделяется количество памяти равное количеству элементов матрицы, умноженному на размер одного элемента в байтах. Для хранения разреженной матрицы выделяется память пропорциональная числу ненулевых элементов матрицы в зависимости от схемы хранения.

Каков принцип обработки разреженной матрицы?

Принцип обработки разреженной матрицы заключается в том, что обрабатываются только ненулевые элементы.

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

Стандартные алгоритмы обработки матриц эффективнее применять при большом проценте ненулевых элементов матрицы (когда нулевые элементы не требуют обработки). Так же стандартная форма хранения более удобна при решении задач, не требующих частых обращений к элементам по их координатам или если матрицы имеют небольшие размеры. В зависимости от размеров матрицы, количества ненулевых элементов и задач выбирается разреженная или стандартная форма хранения матрицы.