|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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

**Отчет по лабораторной работе №3**  
**«Обработка разреженных матриц»**

Выполнил студент:

Исупов Андрей ИУ7-35Б

*2019 г.*

**Описание условия задачи**

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 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:**

1. **Целое число, представляющее собой номер команды:** целое число в диапазоне от **0** до **6**.
2. **Командно-зависимые данные:**
   * количество строк/столбцов матрицы
   * Целые числа подкоманды.

**Выходные данные:**

1. Результирующая матрица в стандартном виде или разреженном виде.
2. Время, затраченное на сложение

**Функция программы:**

0. Закрытие программы с очищением выделенной памяти.

1. Вывод информации о вводе.
2. Автоматическая генерация матриц.
3. Чтение матриц из файлов (в том числе сгенерированных автоматически).
4. Ввод матриц вручную.
5. Вывод матриц на экран.
6. Сложение матриц.

**Обращение к программе:**

Запускается из терминала.

**Аварийные ситуации:**

1. Некорректный ввод номера команды.  
   На входе: число, большее чем 6 или меньшее, чем 0.   
   На выходе: сообщение «Неизвестная команда!»
2. Некорректный ввод количества строк или столбцов матрицы.  
   На входе: неположительное целое число или буква.  
   На выходе: сообщение «Неправильный размер!» и код ошибки.
3. Некорректный ввод элемента матрицы.  
   На входе: число, выходящее за границы условия, или буква.  
   На выходе: сообщение «Ошибка ввода!» и код ошибки.

**Структуры данных**

**Для списка:**

**// Структура узла**

typedef struct node

{

int num; # Значение

struct node \*next; # Указатель на следующий узел

} node\_t;

**// Структура списка**

typedef struct list

{

node\_t \*begin; # Указатель на первый узел

node\_t \*end; # Указатель на последний узел

int length; # Количество элементов

} list\_t;

**Для вектора:**

typedef struct vector\_t

{

int \*nums; # Массив чисел

int length; # Количество элементов

int capacity; # Количество выделенных ячеек памяти

} vector\_t;

**Матрицы:**

**// Структура обычной матрицы**

typedef struct

{

int \*\*M; # Массив матрицы

int m; # Количество строк

int n; # Количество столбцов

} common\_matrix;

**// Структура разреженной матрицы**

typedef struct

{

vector\_t \*A; # Вектор ненулевых элементов

vector\_t \*iA; # Вектор индексов строк ненулевых элементов

list\_t \*jA; # Список индексов первых вхождений ненулевых элементов в стобцах

} sparse\_matrix;

**Функции**

**Interface**

// Вывести на экран название программы

void print\_name\_of\_program();

// Вывести на экран главное меню

void print\_main\_menu();

// Вывести на экран информацию о вводе

void print\_input\_info();

// Вывести на экран интерфейс вывода матриц на экран

void print\_matrix\_output\_interface();

// Вывести на экран интерфейс ввода матриц

void print\_matrix\_input\_interface();

// Вывести на экран интерфейс сложения матриц

void print\_matrix\_menu\_calculate\_interface();

// Вывести на экран интерфейс вывода результата сложения

void print\_matrix\_calculate\_interface();

**List**

// Выделение памяти

void list\_memory\_allocation(list\_t \*l);

// Освобождение памяти

void list\_memory\_deallocation(list\_t \*l);

// Вставка в конец

void list\_push(list\_t \*l, int x);

// Вставка элемента

void list\_insert(list\_t \*l, int index, int x);

// Замена элемента

void list\_change\_element(list\_t \*l, int index, int x);

// Вывод на экран

void list\_print(list\_t \* const l);

**Vector**

// Выделение памяти

void vector\_memory\_allocation(vector\_t \*v);

// Освобождение памяти

void vector\_memory\_deallocation(vector\_t \*v);

// Вывод на экран

void vector\_print(const vector\_t \*v);

// Добавление элемента

void vector\_insert(vector\_t \*v, int index, int x);

// Добавление элемента в конец

void vector\_push\_back(vector\_t \*v, int x);

**Matrix\_calculate**

// Сложение обычных матриц

double common\_matrix\_calculate(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

common\_matrix \*result

);

// Разреженное сложение матриц

double sparse\_matrix\_calculate(

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2,

sparse\_matrix \*sresult

);

// Главная функция суммы матриц

error\_code matrix\_menu\_calculate(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2

);

**Matrix\_io**

// Ввод размера матрицы

error\_code input\_size(int \*x, int \*y);

// Ввод элемента

error\_code input\_element(int \*x);

// Ввод обычной матрицы

error\_code input\_matrix(int x, int y, int \*\*\*matrix);

// Ввод обычных начальных данных

error\_code common\_matrix\_input(common\_matrix \*matrix1, common\_matrix \*matrix2);

// Вывод обычной матрицы на экран

void common\_matrix\_output(common\_matrix \*matrix);

// Ввод разреженных матриц

error\_code a\_ia\_ja\_input(sparse\_matrix \*smatrix, int z, int y, int x);

// Вывод разреженной матрицы

void sparse\_matrix\_output(sparse\_matrix \*smatrix);

// Ввод разреженных начальных данных

error\_code sparse\_matrix\_input(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2

);

// Конвертирование из разреженной матрицы в обычную

void sparse\_in\_common(sparse\_matrix \*smatrix, common\_matrix \*matrix);

// Конвертация из обычной матрицы в разреженную

void common\_in\_sparse(common\_matrix \*matrix, sparse\_matrix \*smatrix);

// Меню ввода

error\_code matrix\_input\_menu(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2

);

// Меню вывода

void matrix\_output\_menu(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2

);

**Matrix\_memory**

// Выделение обычной памяти

int common\_memory\_allocation(int x, int y, int \*\*\*matrix);

// Освобождение обычной матрицы

void common\_memory\_deallocation(int x, int \*\*\*matrix);

// Освобождение разреженной матрицы

void sparse\_memory\_deallocation(sparse\_matrix \*smatrix);

// Инициализация новой разреженной матрицы

void sparse\_memory\_allocation(sparse\_matrix \*smatrix);

**Work\_with\_file**

// Считывание рамеров из файла

error\_code matrix\_read\_size(FILE \*file, common\_matrix \*matrix);

// Считывание элементов из файла

error\_code matrix\_read\_elements(FILE \*file, common\_matrix \*matrix);

// Считывание матрицы из файла

error\_code matrix\_read(common\_matrix \*matrix, char \*name);

// Главная функция считывания из файла

error\_code matrix\_main\_read(

common\_matrix \*matrix1,

common\_matrix \*matrix2,

sparse\_matrix \*smatrix1,

sparse\_matrix \*smatrix2

);

// Генерация случайного числа

int generate\_random\_num();

// Генерация матрицы в файле

error\_code matrix\_create(char \*name);

// Главная функция генерации матрицы в файле

error\_code matrix\_main\_create();

**Алгоритм**

1. Пользователь вводит номер команды из меню.
2. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия с матрицами.
3. При вводе или генерации матрицы, матрица сразу хранится двумя способами хранения: обычном и разреженном.
4. В случае выбора обычного сложения, матрицы складываются поэлементно.
5. В случае выбора разреженного сложения, действия производятся непосредственно над разреженными матрицами. Сравнивается каждый столбец, при этом каждый из них сначала рассматривается как массив построчных вхождений. Если в массивах нет одинаковых вхождений, то в итоговую матрицу элементы записываются по порядку, в порядке возрастания по двум массивам. Если же одинаковые вхождения есть, то соответствующие элементы складываются, после все элементы вновь записываются в порядке возрастания индексов вхождения.

**Тесты**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Ввод | Вывод |
| 1 | Некорректный ввод команды | 7 | Неизвестная команда! |
| 2 | Некорректный ввод количества строк | -1 | Неправильный размер! |
| 3 | Некорректный ввод количества строк | string | Неправильный размер! |
| 4 | Некорректный ввод количества столбцов | -1 | Неправильный размер! |
| 5 | Некорректный ввод количества столбцов | string2 | Неправильный размер! |
| 6 | Некорректный ввод индекса строки | При матрице 4х4: 4 | Недопустимый индекс! |
| 7 | Некорректный ввод индекса строки | strstrstr | Ошибка ввода! |
| 8 | Некорректный ввод индекса столбца | При матрице 3х3: 5 | Недопустимый индекс! |
| 9 | Некорректный ввод индекса столбца | А | Ошибка ввода! |
| 10 | Некорректный ввод элемента матрицы | $ | Ошибка ввода! |

**Оценка эффективности**

**Время сложения в секундах:**

**10% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 0.000005 | 0.000006 |
| 100х100 | 0.000212 | 0.000408 |
| 1000х1000 | 0.016703 | 0.027091 |

**30% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 0.000020 | 0.000030 |
| 100х100 | 0.000240 | 0.000540 |
| 1000х1000 | 0.025728 | 0.035748 |

**50% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 0.000051 | 0.000028 |
| 100х100 | 0.000557 | 0.000520 |
| 1000х1000 | 0.032044 | 0.026781 |

**100% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 0.000060 | 0.000038 |
| 100х100 | 0.001796 | 0.000610 |
| 1000х1000 | 0.073035 | 0.040230 |

**Объём занимаемой памяти (в байтах):**

**10% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 324 | 496 |
| 100х100 | 9 468 | 40 816 |
| 1000х1000 | 1 060 652 | 4 008 016 |

**30% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 452 | 496 |
| 100х100 | 34 044 | 40 816 |
| 1000х1000 | 4 206 380 | 4 008 016 |

**50% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 708 | 496 |
| 100х100 | 66 812 | 40 816 |
| 1000х1000 | 4 208 380 | 4 008 016 |

**100% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 1 220 | 496 |
| 100х100 | 132 348 | 40 816 |
| 1000х1000 | 8 400 682 | 4 008 016 |

**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, столбцовый 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 m\*n ячеек памяти, где m – количество строк, а n – количество столбцов, также массив из m элементов для хранения указателей на строки и один элемент для указателя на саму матрицу. Для разреженной матрицы – зависит от способа. В случае разреженного формата, требуется 3\*k ячеек памяти, где k – количество ненулевых элементов, также 3 указателя на необходимые структуры (векторы и список).

**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, примерно от 30-40%. Также, если не столько важна память, сколько время, то тоже приоритетнее использование обычной матрицы.

**Вывод**

Использование разреженной матрицы выгодно при небольшом количестве ненулевых элементов, примерно до 30-40% от всех элементов. Хранение матрицы в разреженном виде требует большего количества памяти при большом количестве ненулевых элементов, так как нужно хранить достаточно много целочисленных компонентов ненулевых элементов матрицы. Также такой вид хранения требует больше времени на обработку, так как алгоритм сложения достаточно сложен.