|  |  |
| --- | --- |
|  | **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**

Название предмета: Типы и структуры данных

Студент: Гриценко Алексей Михайлович

Группа: ИУ7-31Б

*2020г.*

1. **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.
4. **Техническое задание**
5. ***Описание исходных данных***
6. Целое число, представляющее собой номер команды:целое число в диапазоне от **0** до **6**.
7. Данные при выполнении**:**
   * количество строк/столбцов матрицы, количество элементов матрицы, элементы матрицы в формате **“**индекс строки - индекс столбца – значение элемента**”**;
   * количество строк/столбцов матрицы, процент ненулевых элементов матрицы.
8. ***Описание результата программы***
9. Исходные и результирующая матрицы в стандартном виде или разреженном столбцовом виде.

* Количественная характеристика памяти и времени, затраченного при сложении двух разных видов матриц -- можно лучше

1. ***Задача, реализуемая программой***
2. Ввод матрицы вручную.
3. Генерирование матрицы случайно (по проценту заполненности).
4. Сложение матриц, используя стандартный способ хранения.
5. Сложение матриц, используя разреженный столбцовый вид хранения.
6. Вывод исходных и результирующую матрицы в стандартном виде.
7. Вывод исходных и результирующую матрицы в разреженном столбцовом виде
8. ***Обращения к программе***

Способ обращения к программе происходит через консоль.

1. ***Описание возможных аварийных ситуаций и ошибок пользователя***
2. Некорректный ввод номера команды.  
   На входе: число, большее чем 6 или меньшее, чем 0.   
   На выходе: сообщение «Введена недопустимая команда! Повторите попытку.»
3. Некорректный ввод количества строк или столбцов матрицы.  
   На входе: неположительное целое число или буква.  
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку.»
4. Некорректный ввод индекса строки или столбца матрицы.  
   На входе: число , выходящее за границы луча [0;количество\_столбцов(строк)), или буква.  
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку.»
5. Некорректный ввод элемента матрицы.  
   На входе: число, выходящее за границы условия, или буква.  
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку.»
6. **Описание внутренних структур данных**

За стандартное хранение матрицы отвечает именованная структура **matrix\_t**, описанная как:

typedef struct

{

*int* \*\*matrix;

*int* rows;

*int* columns;

} matrix*\_t***;**

Поля структуры:

* *int* \*\*matrix **–** массив указателей на строки матрицы
* *int* rows ***–***количество строк матрицы;
* *int* columns ***–*** количество столбцов матрицы.

За хранение матрицы в разреженном столбцовом виде отвечает именованная структура **sparse\_t**, описанная как:

typedef struct

{

*int* \*elems;

*int* \*row\_entry;

*int* \*col\_entry;

*int* elems\_amount;

*int* cols\_amount;

} sparse\_t;

Поля структуры:

* *int* \*elems **–** массив элементов матрицы, заполняемый проходом по столбцам;
* *int \**row\_entry ***–***массив, каждый элемент которого равен номеру строки соответствующего элемента из **elems**;
* *int \**col\_entry ***–*** массив, каждый элемент которого указывает на индекс элемента из **elems**, с которого начинается описание столбца;
* *int* elems\_amount **–** количество элементов матрицы;
* *int* cols\_amount **–** количество столбцов матрицы.

1. **Описание алгоритма**
2. Пользователь вводит номер команды из меню.
3. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия с матрицами.
4. При вводе (или генерации) матрицы, матрица сразу хранится двумя способами хранения (стандартном и разреженном столбцовом).
5. В случае выбора стандартного сложения, матрицы складываются в стандартном виде “элемент к элементу”.
6. В случае выбора разреженного столбцового сложения, действия производятся непосредственно над разреженными столбцовыми 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.

**Функции, использующиеся в программе**

void classic\_sum(const matrix\_t matrix\_a, const matrix\_t matrix\_b, matrix\_t \*const matrix\_res, uint64\_t \*const ticks) - функция суммирования матриц в классическом виде, получает их сумму и количество тиков

int sdots(const sparse\_t sparse\_a, const sparse\_t sparse\_b, int \*const dots) – функция подсчета ненулевых точек

int arrays\_min(type\_t \*a\_arr, const int a\_len, type\_t \*b\_arr, const int b\_len) – функция нахождения меньшего из двух массивов

void sparse\_sum(const sparse\_t sparse\_a, const sparse\_t sparse\_b, sparse\_t \*const sparse\_res, uint64\_t \*const ticks) – функция складывания двух матриц в разряженном виде, получает их сумму и количество тиков

void gorandom(matrix\_t \*const matrix, int percent) – рандомизатор

int input\_interval(int \*const num, const int left, const int right) – функция понимания, входит ли число в итервал

int screate(sparse\_t \*const sparse, const int dots, const int cols) – функция создания разреженной матрицы

int sinput(sparse\_t \*const sparse, const matrix\_t matrix) – функция ввода разреженной матрицы

int soutput(const sparse\_t sparse) – функция вывода разреженной матрицы

int create(matrix\_t \*const matrix) – функция создания стандартной матрицы

int input(matrix\_t \*const matrix, const int dots) – функция ввода стандартной матрицы

int output(const matrix\_t \*const matrix) – функция вывода стандартной матрицы

int deletee(matrix\_t \*const matrix) – функция “удаления” стандартной матрицы

int sdelete(sparse\_t \*const sparse) – функция “удаления” разреженной матрицы

1. **Тестирование**

**Негативные тесты:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Некорректный̆ ввод команды | 7 | Введена недопустимая команда! Повторите попытку. |
| 2 | Некорректный̆ ввод количества строк | 0 | Введено недопустимое значение! Повторите попытку |
| 3 | Некорректный ввод количества строк | A | Введено недопустимое значение! Повторите попытку |
| 4 | Некорректный ввод количества столбцов | -2 | Введено недопустимое значение! Повторите попытку |
| 5 | Некорректный ввод количества столбцов | B | Введено недопустимое значение! Повторите попытку |
| 6 | Некорректный ввод индекса строки | При матрице 4х4: 4 | Введено недопустимое значение! Повторите попытку |
| 7 | Некорректный ввод индекса строки | A | Введено недопустимое значение! Повторите попытку |
| 8 | Некорректный ввод индекса столбца | При матрице 3х3: 5 | Введено недопустимое значение! Повторите попытку |
| 9 | Некорректный ввод индекса столбца | C | Введено недопустимое значение! Повторите попытку |
| 10 | Некорректный ввод элемента матрицы | D | Введено недопустимое значение! Повторите попытку |
| 11 | Некорректный ввод элемента матрицы | 2.1 | Введено недопустимое значение! Повторите попытку |

1. **Оценка эффективности**

Измерения эффективности способов сложения будут производиться в единицах измерения – тактах процессов. При записи результатов использовалось среднее количество тактов, полученное по результатам 10 измерений.

**Время сложения:**

**5% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 2428 | 3140 |
| 100х100 | 160336 | 231540 |
| 200х200 | 595094 | 533570 |

**10% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 5780 | 2912 |
| 100х100 | 469980 | 174144 |
| 200х200 | 1738708 | 538044 |

**20% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 10206 | 2758 |
| 100х100 | 945370 | 170288 |
| 200х200 | 5523466 | 565194 |

**30% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 18162 | 3890 |
| 100х100 | 2081624 | 662358 |
| 200х200 | 10924806 | 1544520 |

**40% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 24374 | 2944 |
| 100х100 | 3251860 | 173020 |
| 200х200 | 16944894 | 524542 |

**50% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 32204 | 3416 |
| 100х100 | 4602544 | 193716 |
| 200х200 | 24293226 | 559884 |

**100% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 40380 | 3274 |
| 100х100 | 12498998 | 173600 |
| 200х200 | 69347844 | 620680 |

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

**5% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 152 | 496 |
| 100х100 | 6128 | 40816 |
| 200х200 | 19912 | 161616 |

**10% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 216 | 496 |
| 100х100 | 10880 | 40816 |
| 200х200 | 36112 | 161616 |

**20% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 384 | 496 |
| 100х100 | 20432 | 40816 |
| 200х200 | 70560 | 161616 |

**30% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 504 | 496 |
| 100х100 | 29168 | 40816 |
| 200х200 | 103672 | 161616 |

**40% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 576 | 496 |
| 100х100 | 37792 | 40816 |
| 200х200 | 137056 | 161616 |

**50% заполнения**

|  |  |  |
| --- | --- | --- |
| Размеры | Разреженная матрица | Обычная матрица |
| 10х10 | 752 | 496 |
| 100х100 | 48064 | 40816 |
| 200х200 | 174560 | 161616 |

1. **Выводы по проделанной работе**

Использование алгоритмов хранения и обработки разреженных матриц в случае сложения выгодно при малом количестве ненулевых элементов, примерно до 10% заполненности ,при этом n <=10. Хранение матриц в разреженном столбцовом формате проигрывает по памяти в большинстве случаев, когда процентное заполненение превышает приблизительно 25 %. Помимо занимаемой памяти, разреженные матрицы гораздо дольше обрабатываются практически при любой ситуации. Из этого можно сделать вывод, что если программист работает с разреженной матрицей, и ему важна память, занимаемая программой, ему лучше воспользоваться столбцовым методом хранения. В иных случаях предпочтительнее пользоваться стандартным способом хранения

1. **Ответы на 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 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%). Cтоит отметить, что если не так важна память, занимаемая матрицами, но важно время, то в случае сложения лучше так же воспользоваться стандартными алгоритмами сложения матриц.