Оберган Татьяна ИУ7-35Б

Вариант 18

**Лабораторная работа №3**

Обработка разреженных матриц

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

**Условие задачи (вар 1):**

Разреженная матрица хранится в форме 3-х объектов:

* вектор A содержит значения ненулевых элементов;
* вектор JA содержит номера столбцов для элементов вектора A;
* связный список IA, в элементе Nk которого находится номер компонент в A и JA, с которых начинается описание строки Nk матрицы A.

1. Смоделировать операцию сложения двух матриц, хранящихся в этой форме, с получением результата в той же форме.

2. Произвести операцию сложения.

3. Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц.

**Входные данные:**Две матрицы указанных размеров.

**Выходные данные:**Матрица – результат сложения входных матриц; время, затраченное на сложение.

**Функция программы:**Реализация сложения разреженных матриц; измерение времени на сложение матриц разными способами.

**Возможные ошибки:**

1. Сложение матриц разных размеров
2. Ввод некорректных данных
3. Ввод по неправильному индексу элемента

**Обращение к программе** осуществляется через консоль.

**Структуры данных:**

Матрица в обычном виде:

**int** \*matr1, \*matr2, \*matr3;

Матрица в сложном виде:  
   
**int** \*A1, \*A2, \*A3;  
**int** \*JA1, \*JA2, \*JA3;  
**struct** IA \*IA1 = **NULL**, \*IA2 = **NULL**, \*IA3 = **NULL**, \*tmp;

**struct** IA  
{  
 **int** i;  
 **int** Nk;  
  
 **struct** IA \*next;  
};

**Функции:**

Функции выделения памяти:  
**int** allocate\_matrices(**int** \*\*matr1, **int** \*\*matr2, **int** \*\*matr3, **int** size);  
**void** allocate\_two\_arrays(**int** \*\*a, **int** \*\*b, **int** elements);  
**void** free\_all(**int** \*matr, **int** \*A, **int** \*JA, **struct** IA \*IA);

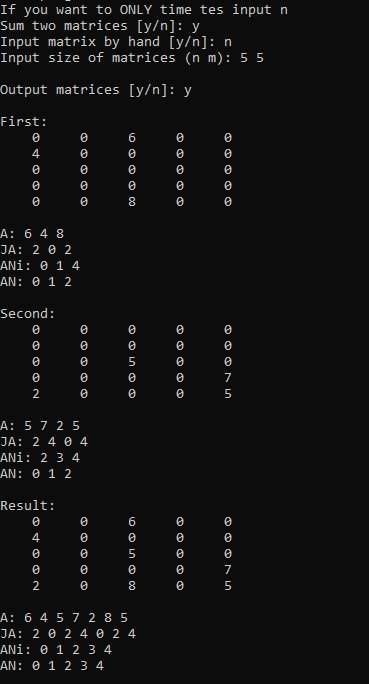
Функции облегчения ввода:  
**int** input\_flag(**char** \*msg);  
**void** input\_size(**char** \*msg, **int** \*n, **int** \*m);

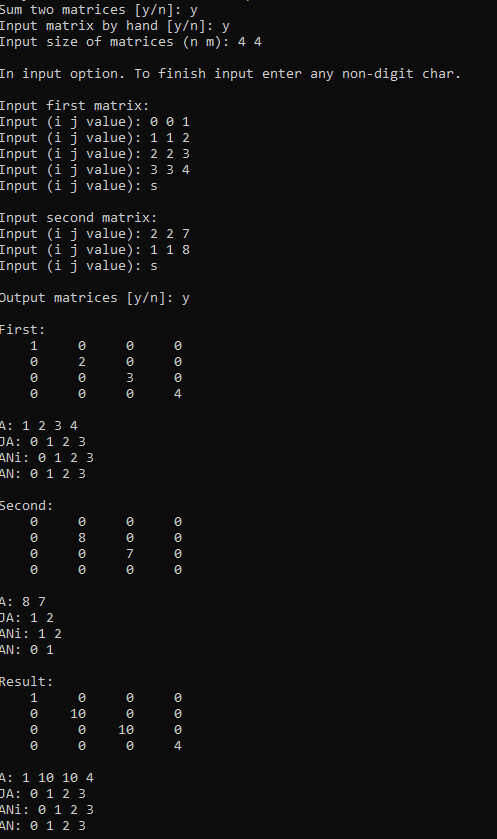
Функции вывода:  
**void** print\_matrix(**const int** \*matr, **int** n, **int** m);  
**void** print\_array(**const int** \*arr, **int** len);  
**void** print\_list(**struct** IA \*IA);  
**void** print\_advanced(**const int** \*A, **const int** \*JA, **struct** IA \*IA, **const int** n\_z\_el);

Функции анализа эффективности: **unsigned long long** tick(**void**);  
**void** time\_testing(**int** n, **int** m, **int** fill);

Функции для работы с матрицами:  
**void** generate\_matrix(**int** \*matr, **int** n, **int** m, **int** fill);  
**void** input\_matrix(**int** \*matr, **int** n, **int** m);  
  
**void** count\_non\_zero(**const int** \*matr, **int** n, **int** m, **int** \*non\_zero\_rows, **int** \*non\_zero\_elements);  
**void** convert\_matrix(**const int** \*matr, **int** n, **int** m, **int** \*A, **int** \*JA, **struct** IA \*IA);  
  
**void** add\_matrices\_simple(**const int** \*matr1, **const int** \*matr2, **int** \*matr3, **int** n, **int** m);  
**void** add\_matrices\_advanced(**const int** \*A1, **const int** \*JA1, **struct** IA \*IA1, **int** n\_z\_el1,  
 **const int** \*A2, **const int** \*JA2, **struct** IA \*IA2, **int** n\_z\_el2,  
 **int** \*A3, **int** \*JA3, **struct** IA \*IA3, **int** \*n\_z\_el3);  
  
**void** copy\_row(**const int** \*A1, **const int** \*JA1, **struct** IA \*IA1, **int** max\_iel,  
 **int** \*A3, **int** \*JA3, **struct** IA \*IA3, **int** \*n\_z\_el3);  
**void** add\_rows(**const int** \*A1, **const int** \*JA1, **struct** IA \*IA1, **int** max\_i1el,  
 **const int** \*A2, **const int** \*JA2, **struct** IA \*IA2, **int** max\_i2el,  
 **int** \*A3, **int** \*JA3, **struct** IA \*IA3, **int** \*n\_z\_el3);

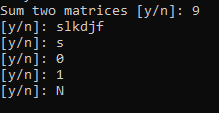
**Интерфейс:**Матрицу можно вводить как вручную или сгенерировать автоматически.



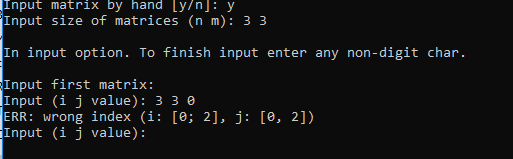


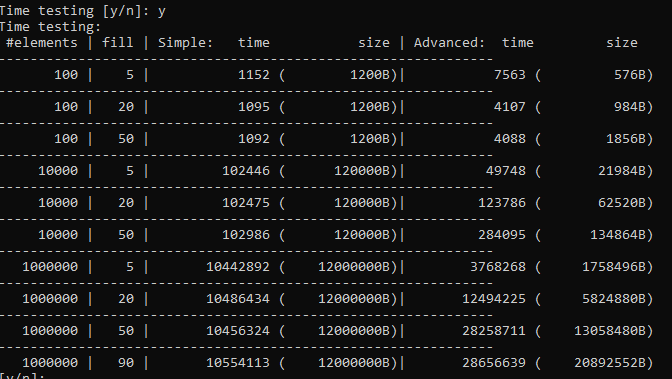
**Тесты:**

1) обработка ответа на y/n вопрос



2) доступ по некорректному индексу



**Анализ эффективности (по памяти и времени):**  


Операция сложения над матрицами с заполнением до 10% элементов выполняется быстрее при хранении ее как разреженной. При этом требуется в 6 раз меньше памяти.  
  
Операция сложения над матрицами с 20% элементов выполняется медленнее в 1.5 раза при хранении в специальном виде. При этом подходе используется в 2 раза меньше памяти.   
  
Хранение матрицы с заполнением в 50% в разреженном виде требует на 10% больше памяти, чем в обычном. Также время сложения в данном виде будет в 2 раза медленнее.

**Контрольные вопросы:**

**1)** Что такое разреженная матрица, какие схемы хранения таких матриц вы знаете?

Разреженная матрица – матр., большая часть которой заполнена нулями.

Можно хранить индекс (i,j) ненулевого элемента и его значение .

Можно хранить значения ненулевых элементов, их индексы строки/столбца, номера элементов, с которых начинается очередная строка.

**2)** Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы.  
Под обычную матрицу выделяется n\*m\*sizeof(element) байт памяти.   
Требуемая память под хранение разреженной матрицы зависит от выбранного типа хранения, количества ненулевых элементов.

**3)** Каков принцип обработки разреженной матрицы?

Обрабатываются только ненулевые элементы, что позволяет сократить время обработки.

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

Эффективнее применять стандартные алгоритмы обработки матриц при достижении определенного уровня заполненности матрицы.

**Вывод:**

Алгоритмы работы с разреженными матрицами дают заметный выигрыш во времени при обработке достаточно больших матриц с малым заполнением.