**Министерство образования науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

«Информационные технологии и прикладная математика»

**Курсовой проект**

**По курсу «Практикум на ЭВМ»**

**2 семестр**

**Задание 6:**

«Разреженные матрицы»

|  |  |
| --- | --- |
| Группа: | М8О-106Б-21 |
| Студент: | Орусский В.Р. |
| Преподаватель: | Дубинин А.В. |
| Оценка: |  |
| Дата: |  |

**Оглавление**

[Введение 3](#_Toc104744625)

[Постановка задачи 4](#_Toc104744626)

[Вариант 9: 4](#_Toc104744627)

[Теоретическая часть 4](#_Toc104744628)

[Описание алгоритма 5](#_Toc104744629)

[Использованные в программе переменные 6](#_Toc104744630)

[Пользовательские структуры данных в программе: 6](#_Toc104744631)

[Исходники программы 7](#_Toc104744632)

[Протокол исполнения и тесты 11](#_Toc104744633)

[Вывод 12](#_Toc104744634)

# Введение

Мы будем работать с разреженными матрицами (матрицы, где подавляющее большинство элементов – нулевые). В данном задании нам потребуется изучить различные варианты представления матриц с помощью стандартных структур данных (с 1 / 2 / 3 векторов и так далее). Также нам надо будет выполнить над разреженными матрицами различные преобразования.

# Постановка задачи

Составить программу на языке Си с процедурами и/или функциями для обработки прямоугольных разреженных матриц, которая:

1) Вводит матрицы различного размера, представленные во входном текстовом файле в обычном формате с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой

2) Печатает введённые матрицы во внутреннем представлении согласно заданной схеме размещения и в обычном (естественном виде)

3) Выполняет необходимые преобразования разреженных матриц (или вычисления над ними) путём обращения к соответствующим процедурам и/или функциям

4) Печатает результат преобразования согласно заданной схеме размещения и в обычном виде

Изучить различные виды представления матриц с помощью различных структур в СП Си.

## Вариант 9:

Найти столбец, содержащий наибольшее количество ненулевых элементов, и напечатать его номер и произведение элементов этого столбца. Если таких столбцов несколько, обработать предпоследний

# Теоретическая часть

Разреженные матрицы – это матрица с преимущественно нулевыми элементами. В противном случае (если большинство элементов ненулевые), то матрица называется плотной.

Но не всё так просто, разреженные матрицы интересны нам из-за своих размеров. Порой, например, при решении таких задач, как дифференциальное уравнение в частных производных, разреженные матрицы могут достигать огромных размеров (1000 x 1000). И представить такую матрицу в стандартном виде в ПК, тем более на бумаге – проблематично, а заниматься их перемножением или любыми другими операциями невероятно долго и ресурсоёмко. Для этого и было придумано представление разреженных матриц только в виде хранения их ненулевых элементов. Поскольку, вместо 1 МЛН элементов мы будем хранить около 0.1% (примерно столько ненулевых элементов в разреженных матрицах), то мы будем экономить огромное кол-во памяти и времени при работе.

Существует несколько способов хранения, которые отличаются своим удобством для изменения структуры, либо скоростью доступа к данным и так далее:

Словарь по ключам (DOK - Dictionary of Keys) строится как словарь, где ключ — это пара (строка, столбец), а значение это соответствующий строке и столбцу элемент матрицы.

Список списков (LIL - List of Lists) строится как список строк, где строка — это список узлов вида (столбец, значение).

Список координат (COO - Coordinate list) хранится список из элементов вида (строка, столбец, значение).

Все эти реализации предусмотрены и в нашей работе.

# Описание алгоритма

Для начала были реализована структура вектора (vector.c и vector.h), которые обладают всеми необходимыми функциями для правильного функционирования вектора.

После этого была реализована функция «vectorFill», которая заполняет три вектора (CIP, PI, YE) значениями в зависимости от тех значений, что были получены прочтением из файла.

После была написана функция вывода матрицы в стандартном виде (matrixPrint).

А уже в самом main программы была реализована функция нахождения двух максимумов по кол-ву ненулевых элементов (второй максимум нужен, чтобы найти предпоследний столбец). И подсчёт произведения элементов данного столбца.

# Использованные в программе переменные

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | За что отвечает |
| num | integer | Числовое значение элемента матрицы |
| file | FILE\* | Файл с матрицей |
| CIP | Vector | Вектор хранящий индекс начала строки в векторах PI YE |
| PI | Vector | Вектор, хранящий номера столбцов |
| YE | Vector | Вектор, хранящий значения элементов |
| PIi | integer | Индекс вектора PI |
| CIPi | integer | Индекс вектора CIP |
| rows, col | integer | Размеры матрицы |
| c | char | Действие в текстовом меню |
| v1 | Vector | Вектор для хранения значений столбца |
| max | integer | Хранение численного значения максимума |
| max\_i1(2) | integer | Хранение индексов двух максимумов |
| flag | integer | Указатель на то, было ли два максимума или нет |
| result | integer | Результат умножения |

# Пользовательские структуры данных в программе:

|  |  |  |
| --- | --- | --- |
| Название | Поля | Значения |
| Vector | int \_data | Указатель на начало массива данных в векторе |
| int \_size; | Размер вектора |
| int \_capacity | Размер буфера |

# Исходники программы

Файл «vector.h» – заголовочный файл, в котором объявлена структура вектора

#ifndef **VECTOR\_H**#define **VECTOR\_H**#include <stdlib.h>  
  
*// Структура вектора на integer  
typedef struct* \_Vector {  
 *int* \*\_data;  
 *int* \_size;  
 *int* \_capacity;  
} Vector;  
  
Vector \*VectorCreate(*const int* size);  
*int* VectorEmpty(*const* Vector \*v);  
*int* VectorSize(*const* Vector \*v);  
*int* VectorCapacity(*const* Vector \*v);  
*int* VectorIndex(*const* Vector \*v, *const int* index);  
*int* VectorPushBack(Vector \*v, *const int* value);  
*void* VectorResize(Vector \*v, *const int* size);  
*int* VectorEqual(*const* Vector \*v1, *const* Vector \*v2);  
*void* VectorDestroy(Vector \*v);  
*void* VectorPrint(Vector \*v, *const int* size);  
  
#endif

Файл «vector.c» - реализация указанных выше функций.

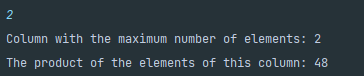
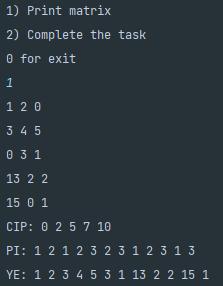
#include <stdio.h>  
#include "vector.h"  
  
*// Создаём вектор*Vector \*VectorCreate(*const int* size) {  
 Vector \*v;  
 v = (Vector \*)malloc(*sizeof*(Vector));  
 *if* (size > 0) {  
 v->\_data = (*int* \*)malloc(*sizeof*(*int*) \* size);  
 v->\_capacity = size;  
 } *else* {  
 v->\_data = (*int* \*)malloc(*sizeof*(*int*));  
 v->\_capacity = 1;  
 }  
 v->\_size = 0;  
}  
  
*// Является ли вектор пустым  
int* VectorEmpty(*const* Vector \*v) {  
 *return* v->\_size == 0;  
}  
  
*// Размер вектора  
int* VectorSize(*const* Vector \*v) {  
 *return* v->\_size;  
}  
  
*int* VectorCapacity(*const* Vector \*v) {  
 *return* v->\_capacity;  
}  
  
*//Получаем элемент вектора по индексу  
int* VectorIndex(*const* Vector \*v, *const int* index) {  
 *return* v->\_data[index];  
}  
  
*// Добавляем элемент к вектору с конца  
int* VectorPushBack(Vector \*v, *const int* value) {  
 *int* \*ptr = **NULL**;  
 *if* (v->\_size == v->\_capacity) {  
 ptr = (*int* \*)realloc(v->\_data, *sizeof*(*int*) \* v->\_capacity \* 2);  
 *if* (ptr != **NULL**) {  
 v->\_data = ptr;  
 v->\_capacity \*= 2;  
 } *else  
 return* 0;  
 }  
 v->\_data[v->\_size++] = value;  
 *return* 1;  
}  
  
*// Изменяем размер вектора  
void* VectorResize(Vector \*v, *const int* size) {  
 *int* \*ptr = **NULL**;  
 *if* (size < 0)  
 *return*;  
 *if* (size == 0) {  
 VectorDestroy(v);  
 *return*;  
 }  
 ptr = (*int* \*)realloc(v->\_data, *sizeof*(*int*) \* size);  
 *if* (ptr != **NULL**) {  
 v->\_data = ptr;  
 v->\_size = size;  
 v->\_capacity = size;  
 }  
}  
  
*// Сравнение двух векторов  
int* VectorEqual(*const* Vector \*v1, *const* Vector \*v2) {  
 *int* i;  
 *if* (v1->\_size != v2->\_size)  
 *return* 0;  
 *for* (i = 0; i < v1->\_size; i++)  
 *if* (v1->\_data[i] != v2->\_data[i])  
 *return* 0;  
 *return* 1;  
}  
  
*// Уничтожение вектора  
void* VectorDestroy(Vector \*v) {  
 *if* (v->\_data != **NULL**) {  
 free(v->\_data);  
  
 v->\_data = **NULL**;  
 }  
 v->\_size = 0;  
 v->\_capacity = 0;  
}  
  
*// Вывод вектора  
void* VectorPrint(Vector \*v, *const int* size) {  
 *for* (*int* i = 0; i < size; i++)  
 printf("%d ", v->\_data[i]);  
 printf("\n");  
}

Файл «main.c» - файл для взаимодействия с пользователем и реализация функций, выполняющих основное поставленное задание.

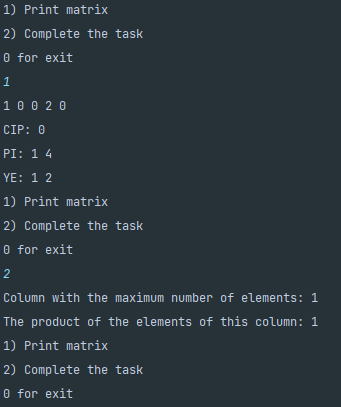
#include <stdio.h>  
#include <stdlib.h>  
#include "vector.h"  
  
*// Текстовое меню для пользователя  
void* print\_menu() {  
 printf("1) Print matrix\n");  
 printf("2) Complete the task\n0 for exit\n");  
}  
  
*// Заполняем вектор  
void* vectorFill(Vector \*\*CIP, Vector \*\*PI, Vector \*\*YE, FILE \*file, *int* rows, *int* col) {  
 *int* num;  
 *if* (file) {  
 *int* count = 0;  
 *// Очищаем память, если был передан не пустой вектор  
 if* (\*CIP) free(\*CIP);  
 *if* (\*PI) free(\*PI);  
 *if* (\*YE) free(\*YE);  
 *// Заполняем векторы значениями по умолчанию* \*CIP = VectorCreate(1);  
 \*PI = VectorCreate(1);  
 \*YE = VectorCreate(1);  
 *// Идём по вектору CIP и заполняем у него значения для начала каждой строки  
 for* (*int* i = 1; i <= rows; i++) {  
 VectorPushBack(\*CIP, count);  
 *// Сразу заполняем векторы PI и YE номером столбца и значением ячейки соответственно  
 for* (*int* j = 1; j <= col; j++) {  
 fscanf(file, "%d", &num);  
 *if* (num) {  
 VectorPushBack(\*PI, j);  
 VectorPushBack(\*YE, num);  
 count++;  
 }  
 }  
 }  
 fclose(file);  
 }  
}  
  
*// Вывод матрицы  
void* matrixPrint(Vector \*CIP, Vector \*PI, Vector \*YE, *int* rows, *int* col) {  
 *int* PIi = 0;  
 *int* CIPi = 1;  
 *if* (CIP && PI && YE) {  
 *for* (*int* i = 0; i < VectorSize(CIP); i++) {  
 *for* (*int* j = 1; j <= col; j++) {  
 *if* (PI->\_data[PIi] == j) {  
 printf("%d ", YE->\_data[PIi]);  
 PIi++;  
 } *else* printf("0 ");  
 *if* (CIP->\_data[CIPi] == PIi) {  
 CIPi++;  
 *for* (*int* k = 0; k < col - j; k++)  
 printf("0 ");  
 printf("\n");  
 *break*;  
 }  
 }  
 }  
 }  
 printf("\n");  
}  
  
*int* main(*void*) {  
 *char* c;  
 *int* rows, col;  
 FILE \*file;  
 Vector \*CIP = **NULL**, \*PI = **NULL**, \*YE = **NULL**;  
 Vector \*v1 = **NULL**;  
 v1 = VectorCreate(1);  
 print\_menu();  
 *while* ((c = getchar()) != **EOF**) {  
 *if* (c == '\n') *continue*;  
 file = fopen("matrix.txt", "r");  
 *if* (file) {  
 fscanf(file, "%d%d", &rows, &col);  
 vectorFill(&CIP, &PI, &YE, file, rows, col);  
 *switch* (c) {  
 *case* '1': matrixPrint(CIP, PI, YE, rows, col);  
 printf("CIP: ");  
 VectorPrint(CIP, VectorSize(CIP));  
 printf("PI: ");  
 VectorPrint(PI, VectorSize(PI));  
 printf("YE: ");  
 VectorPrint(YE, VectorSize(YE));  
 *break*;  
 *case* '2': VectorResize(v1, col + 1);  
 *for* (*int* i = 0; i < VectorSize(v1); i++)  
 v1->\_data[i] = 0;  
 *for* (*int* i = 0; i < VectorSize(PI); i++)  
 v1->\_data[PI->\_data[i]]++;  
 *int* max = 0, max\_i1 = 0;  
 *int* max\_i2 = 0;  
 *int* flag = 0;  
 *for* (*int* i = 1; i < VectorSize(v1); i++)  
 *if* (v1->\_data[i] > max) {  
 max = v1->\_data[i];  
 max\_i1 = i;  
 flag = 0;  
 } *else if* (v1->\_data[i] == max) {  
 max\_i2 = max\_i1;  
 max\_i1 = i;  
 flag = 1;  
 }  
 *if* (flag) max\_i1 = max\_i2;  
 printf("Column with the maximum number of elements: %d\n", max\_i1);  
 *int* result = 1;  
 *for* (*int* i = 0; i < VectorSize(PI); i++)  
 *if* (PI->\_data[i] == max\_i1) result \*= YE->\_data[i];  
 printf("The product of the elements of this column: %d\n", result);  
 *break*;  
 *case* '0': VectorDestroy(CIP);  
 VectorDestroy(PI);  
 VectorDestroy(YE);  
 VectorDestroy(v1);  
 *return* 0;  
 *default*: printf("There is no such option! I guess you made a mistake : (\nGoodbye!\n");  
 *return* 0;  
 }  
 print\_menu();  
 } *else* printf("File isn't exist!\n");  
 }  
}

# Протокол исполнения и тесты

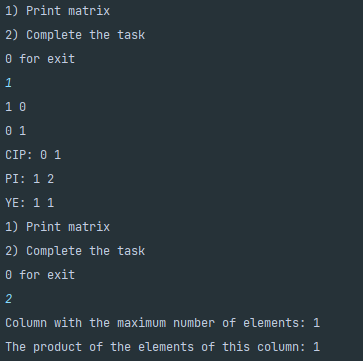
Тест №1



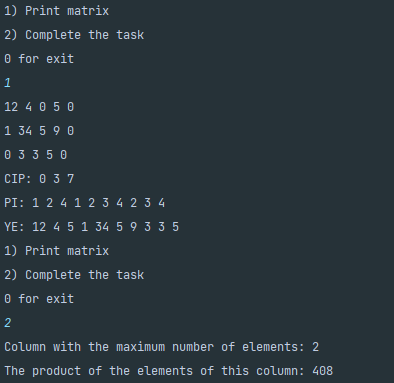
Тест №2



Тест №3



Тест №4



# Вывод

Для выполнения данной работы был изучен процесс представления матриц огромных размеров в ПК с целью экономии памяти и времени по работе с ними. Данная работа хорошо развивает мышление и нестандартный подход к ситуации.

Знания будут полезны при работе с графами, использовании различных алгоритмов для разрешения дифференциальных уравнений (метод конечных элементов), для анализа больших статистических данных, где есть много нулевых значений.