Московский Авиационный Институт
(Национально Исследовательский Университет)
Факультет информационных технологий и прикладной математики

Курсовая работа по курсу
«Практикум на ЭВМ»
Второй семестр

Задание 7

«Разреженные матрицы»

Группа:	М8О-107Б-20
Студент:	Алапанова Эльза Халилевна
Преподаватель:	Найденов Иван Евгеньевич
Оценка:	
Дата:	

Цель работы:

Составить программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами целого типа, которая:

- Вводит матрицы различного размера с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой;
- Печатает введенные матрицы во внутреннем представлении и в обычном виде;
- Выполняет необходимые преобразования разреженных матриц (или вычисления над ними) путем обращения к соответствующим функциям;
- Печатает результат преобразования во внутреннем представлении и в обычном виде.

В процедурах и функциях предусмотреть проверки и печать сообщений в случаях ошибок в задании параметров. Для отладки использовать матрицы, содержащие 5-10% ненулевых элементов, с максимальным числом элементов 100.

Задание:

Вариант 2

Задание:
Определить максимальный по модулю элемент матрицы и разделить на него все элементы столбца, в котором он находится. Если таких элементов несколько, обработать предпоследний столбец, содержащий такой элемент.

Схема размещения матрицы:

2. Один вектор:

Ненулевому элементу соответствуют две ячейки: первая содержит номер столбца, вторая содержит значение элемента. Нуль в первой ячейке означает конец строки, а вторая ячейка содержит в этом случае номер следующей хранимой строки. Нули в обеих ячейках являются признаком конца перечня ненулевых элементов разреженной матрицы.

0	Номер строки	Номер столбца	Значение	Номер столбца	Значение	...

...

0	Номер строки	Номер столбца	Значение	...	0	0

Теоретическая часть:

Разрежённая матрица — это матрица с преимущественно нулевыми элементами. В противном случае, если бо́льшая часть элементов матрицы ненулевые, матрица считается плотной.
При хранении и преобразовании разрежённых матриц в компьютере бывает полезно, а часто и необходимо, использовать специальные алгоритмы и структуры данных, которые учитывают разрежённую структуру матрицы. Операции и алгоритмы, применяемые для работы с обычными, плотными матрицами, применительно к большим разрежённым матрицам работают относительно медленно и требуют значительных объёмов памяти. Однако разрежённые матрицы могут быть легко сжаты путём записи только своих ненулевых элементов, что снижает требования к компьютерной памяти.

Метод решения

Проект состоит из трех частей: основной файл kp7.c, а также файлы vector.c, vector.h для реализации динамической структуры данных -
вектора.

Код программы:

Файл vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <stdlib.h>

typedef struct _Comp
{
	double a;
	double b;
} Comp;

typedef struct _Item
```

```
{
        int ind;
        Comp c;
} Item;

typedef Item VECTOR_TYPE;

typedef struct _Vector
{
        VECTOR_TYPE *_data;
        int _size;
        int _capacity;
} Vector;

void vectorCreate(Vector *v, const int size);
int vectorEmpty(const Vector *v);
int vectorSize(const Vector *v);
int vectorCapacity(const Vector *v);
VECTOR_TYPE vectorLoad(const Vector *v, const int index);
void vectorSave(Vector *v, const int index, const VECTOR_TYPE value);
int vectorPushBack(Vector *v, const VECTOR_TYPE value);
void vectorResize(Vector *v, const int size);
//int vectorEqual(const Vector *v1, const Vector *v2);
void vectorDestroy(Vector *v);

#endif
```

Файл vector.c

```
#include "vector.h"

void vectorCreate(Vector *v, const int size)
{
   if (size > 0)
   {
      v->_data = (VECTOR_TYPE *)malloc(sizeof(VECTOR_TYPE) * size);
      v->_capacity = size;
   }
   else
   {
      v->_data = (VECTOR_TYPE *)malloc(sizeof(VECTOR_TYPE));
      v->_capacity = 1;
   }

   v->_size = 0;
}

int vectorEmpty(const Vector *v)
{
   return v->_size == 0;
```

```
}

int vectorSize(const Vector *v)
{
   return v->_size;
}

int vectorCapacity(const Vector *v)
{
   return v->_capacity;
}

VECTOR_TYPE vectorLoad(const Vector *v, const int index)
{
   return v->_data[index];
}

void vectorSave(Vector *v, const int index, const VECTOR_TYPE value)
{
   v->_data[index] = value;
}

int vectorPushBack(Vector *v, const VECTOR_TYPE value)
{
   VECTOR_TYPE *ptr = NULL;

   if (v->_size == v->_capacity)
   {
      ptr = (VECTOR_TYPE *)realloc(v->_data, sizeof(VECTOR_TYPE) * v->_capacity * 2);

      if (ptr != NULL)
      {
         v->_data = ptr;
         v->_capacity *= 2;
      }
      else
         return 0;
   }

   v->_data[v->_size++] = value;

   return 1;
}

void vectorResize(Vector *v, const int size)
{
   VECTOR_TYPE *ptr = NULL;

   if (size < 0)
      return;
```

```
    if (size == 0)
    {
        vectorDestroy(v);

        return;
    }

    ptr = (VECTOR_TYPE *)realloc(v->_data, sizeof(VECTOR_TYPE) * size);

    if (ptr != NULL)
    {
        v->_data = ptr;
        v->_size = size;
        v->_capacity = size;
    }
}
/*
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;
}
*/
void vectorDestroy(Vector *v)
{
    if (v->_data != NULL)
    {
        free(v->_data);

        v->_data = NULL;
    }

    v->_size = 0;
    v->_capacity = 0;
}
```

Файл kp7.c

```
#include <stdio.h>
#include <math.h>
```

```
#include "vector.h"

typedef enum _kInd
{
	END = -3,
	COMP,
	EMPTY
} kInd;

typedef struct _Cell
{
	Vector *v;
	int ind;
	int row;
	int col;
	Comp data;
} Cell;

double complexModule(const Comp c);
Comp complexDivide(const Comp c1, const Comp c2);

Cell cellFirst(Vector *v);
void cellNext(Cell *cell);

void printSourceMatrix(Vector *v, const int m, const int n);
void printInnerMatrix(const Vector *v);

int main(void)
{
	const int N = 100;
	int m, n, i, j, isRowBegin, lastInd, cnt, maxCols[N];
	Vector v;
	Comp tmpComp, maxComp;
	Item tmpItem;
	Cell cell;

	for (i = 0; i < N; i++)
		maxCols[i] = 0;

	printf("Введите количество строк: ");
	scanf("%d", &m);
	printf("Введите количество столбцов: ");
	scanf("%d", &n);

	if (m < 1 || m > N)
	{
		printf("Количество строк должно быть в диапозоне от 1 до %d\n", N);

		return 0;
	}
```

```
if (n < 1 || n > N)
{
	printf("Количество столбцов должно быть в диапозоне от 1 до %d\n", N);

	return 0;
}

vectorCreate(&v, 1);

tmpItem.ind = EMPTY;

vectorPushBack(&v, tmpItem);

for (i = 0; i < m; i++)
{
	isRowBegin = 0;

	for (j = 0; j < n; j++)
	{
		printf("Введите действительную и мнимую части ячейки [%d][%d]: ", i, j);
		scanf("%lf %lf", &tmpComp.a, &tmpComp.b);

		if (tmpComp.a == 0.0 && tmpComp.b == 0.0)
			continue;

		if (!isRowBegin)
		{
			isRowBegin = 1;

			tmpItem.ind = i;

			vectorPushBack(&v, tmpItem);
		}

		tmpItem.ind = j;

		vectorPushBack(&v, tmpItem);

		tmpItem.c = tmpComp;
		tmpItem.ind = COMP;

		vectorPushBack(&v, tmpItem);
	}

	if (isRowBegin)
	{
		tmpItem.ind = EMPTY;

		vectorPushBack(&v, tmpItem);
```

```
            }
        }

        tmpItem.ind = END;

        vectorPushBack(&v, tmpItem);

        printf("Обычное представление:\n");
        printSourceMatrix(&v, m, n);
        printf("Внутреннее представление\n");
        printInnerMatrix(&v);

        maxComp.a = 0.0;
        maxComp.b = 0.0;

        cell = cellFirst(&v);

        while (cell.row != END)
        {
            if (complexModule(cell.data) > complexModule(maxComp))
                maxComp = cell.data;

            cellNext(&cell);
        }

        printf("Максимальное комплексное число по модулю: (%.2lf, %.2lf), модуль равен: %.2lf\
n", maxComp.a, maxComp.b, complexModule(maxComp));

        if (maxComp.a == 0.0 && maxComp.b == 0)
        {
            printf("Делить на него нельзя, так как его модуль равен нулю\n");

            return 0;
        }

        lastInd = 0;
        cnt = 0;

        cell = cellFirst(&v);

        while (cell.row != END)
        {
            if (complexModule(cell.data) == complexModule(maxComp))
            {
                maxCols[cell.col] = 1;
                lastInd = cell.col;
                cnt++;
            }

            cellNext(&cell);
```

```
	}

	if (cnt > 1)
		for (i = lastInd - 1; i >= 0; i--)
			if (maxCols[i])
			{
				lastInd = i;

				break;
			}

	cell = cellFirst(&v);

	while (cell.row != END)
	{
		if (cell.col == lastInd)
		{
			tmpItem = vectorLoad(&v, cell.ind + 1);
			tmpItem.c = complexDivide(cell.data, maxComp);

			vectorSave(&v, cell.ind + 1, tmpItem);
		}

		cellNext(&cell);
	}

	printf("Обычное представление после преобразования:\n");
	printSourceMatrix(&v, m, n);
	printf("Внутреннее представление после преобразования:\n");
	printInnerMatrix(&v);

	vectorDestroy(&v);

	return 0;
}

double complexModule(const Comp c)
{
	return sqrt(pow(c.a, 2.0) + pow(c.b, 2.0));
}

Comp complexDivide(const Comp c1, const Comp c2)
{
	const double znam = pow(c2.a, 2.0) + pow(c2.b, 2.0);
	Comp res;

	res.a = (double)(c1.a * c2.a + c1.b * c2.b) / znam;
	res.b = (double)(c2.a * c1.b - c2.b * c1.a) / znam;

	return res;
```

```
}

Cell cellFirst(Vector *v)
{
	Cell res;

	res.v = v;
	res.ind = 2;
	res.row = END;
	res.col = EMPTY;
	res.data.a = 0.0;
	res.data.b = 0.0;

	if (vectorLoad(v, res.ind - 1).ind != END)
	{
		res.row = vectorLoad(v, res.ind - 1).ind;
		res.col = vectorLoad(v, res.ind).ind;
		res.data = vectorLoad(v, res.ind + 1).c;
	}

	return res;
}

void cellNext(Cell *cell)
{
	int c1, c2;

	if (cell->row == END)
		return;

	cell->ind += 2;
	c1 = vectorLoad(cell->v, cell->ind).ind;
	c2 = vectorLoad(cell->v, cell->ind + 1).ind;

	if (c1 > EMPTY && c2 == COMP)
	{
		cell->col = vectorLoad(cell->v, cell->ind).ind;
		cell->data = vectorLoad(cell->v, cell->ind + 1).c;
	}
	else if (c1 == EMPTY && c2 > EMPTY)
	{
		cell->row = vectorLoad(cell->v, cell->ind + 1).ind;
		cell->col = vectorLoad(cell->v, cell->ind + 2).ind;
		cell->data = vectorLoad(cell->v, cell->ind + 3).c;
		cell->ind += 2;
	}
	else
	{
		cell->row = END;
		cell->col = EMPTY;
```

```
	}
}

void printSourceMatrix(Vector *v, const int m, const int n)
{
	int i, j;
	Cell cell = cellFirst(v);

	for (i = 0; i < m; i++)
	{
		for (j = 0; j < n; j++)
		{
			if (i == cell.row && j == cell.col)
			{
				printf("(%.2lf, %.2lf) ", cell.data.a, cell.data.b);

				cellNext(&cell);
			}
			else
				printf("(%.2lf, %.2lf) ", 0.0, 0.0);
		}

		printf("\n");
	}
}

void printInnerMatrix(const Vector *v)
{
	int i;
	Item item;

	for (i = 0; i < vectorSize(v); i++)
	{
		item = vectorLoad(v, i);

		if (item.ind == COMP)
			printf("(%.2lf, %.2lf) ", item.c.a, item.c.b);
		else
			printf("%d ", item.ind);
	}

	printf("\n");
}
```

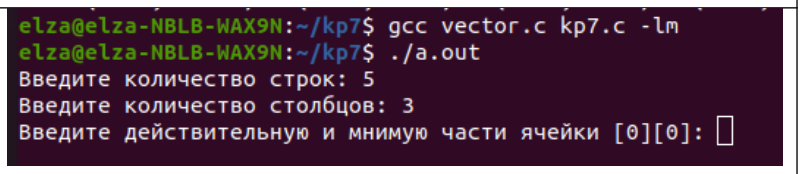
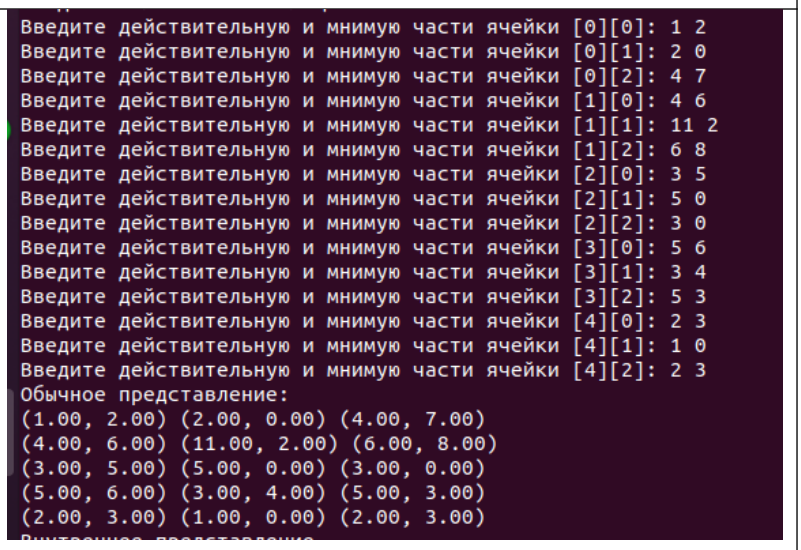
Входные данные

Матрица с действительной и мнимой частью.

Выходные данные

Внутреннее представление матрицы, максимальное комплексное число по модулю, модуль данного числа, обычное представление после преобразования, внутреннее представление после преобразования.

Выполнение:

Что делаем	Терминал
Скомпилируем и введем количество строк и столбцов.	
Введем действительную и мнимую часть матрицы.	
Итог, который получаем	Внутреннее представление -1 0 0 (1.00, 2.00) 1 (2.00, 0.00) 2 (4.00, 7.00) -1 1 0 (4.00, 6.00) 1 (11.00, 2.00) 2 (6.00, 8.00) -1 2 0 (3.00, 5.00) 1 (5.00, 0.00) 2 (3.00, 0.00) -1 3 0 (5.00, 6.00) 1 (3.00, 4.00) 2 (5.00, 3.00) -1 4 0 (2.00, 3.00) 1 (1.00, 0.00) 2 (2.00, 3.00) -1 -3 Максимальное комплексное число по модулю: (11.00, 2.00), модуль равен: 11.18 Обычное представление после преобразования: (1.00, 2.00) (0.18, -0.03) (4.00, 7.00) (4.00, 6.00) (1.00, 0.00) (6.00, 8.00) (3.00, 5.00) (0.44, -0.08) (3.00, 0.00)

	(5.00, 6.00) (0.33, 0.30) (5.00, 3.00) (2.00, 3.00) (0.09, -0.02) (2.00, 3.00) Внутреннее представление после преобразования: -1 0 0 (1.00, 2.00) 1 (0.18, -0.03) 2 (4.00, 7.00) -1 1 0 (4.00, 6.00) 1 (1.00, 0.00) 2 (6.00, 8.00) -1 2 0 (3.00, 5.00) 1 (0.44, -0.08) 2 (3.00, 0.00) -1 3 0 (5.00, 6.00) 1 (0.33, 0.30) 2 (5.00, 3.00) -1 4 0 (2.00, 3.00) 1 (0.09, -0.02) 2 (2.00, 3.00) -1 -3

Вывод

Данная курсовая работа мне понравилась.

Я узнала, что такое разреженные матрицы, научилась задавать их с помощью вектора, а так же работать с ними, писать различные функции для их обработки.

Думаю, данные знания пригодятся мне в будущем при написании какой-либо работы.