Московский Авиационный Институт
(Национальный Исследовательский Институт)

Факультет прикладной математики

КУРСОВАЯ РАБОТА

по курсам
«Инструментальные средства UNIX, алгоритмы и структуры данных»

Задание VII
«Разреженные матрицы»

Студент: Пивницкий Д.
Группа: М8о-101Б-19
Преподаватель: Сорокин С.
Оценка: _____
Дата: _____

Москва, 2019

Оглавление

Введение

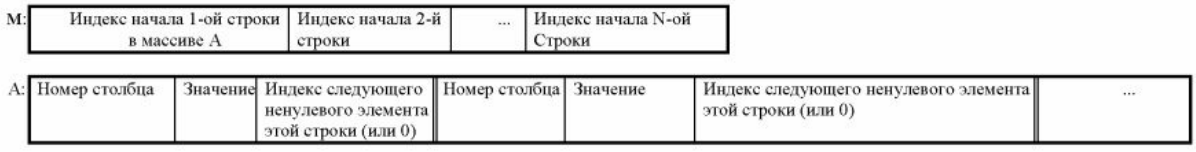
Составить программу на языке Си с процедурами и/или функциями для обработки *прямоугольных* разреженных матриц с элементами комплексного типа, которая:

1. Вводит матрицы различного размера, представленные во входном текстовом файле в обычном формате (по строкам), с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой;
2. Печатает введённые матрицы во внутреннем представлении согласовано 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 (вычисления) согласно заданной схеме размещения 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.

Вариант схемы размещения матрицы: 1

Цепочка ненулевых элементов в векторе *A* со строчным индексированием



Индекс, равный нулю, означает отсутствие ненулевых элементов в строке

Вариант преобразований: 1

Определить максимальный комплексный элемент матрицы по модулю и разделить на него все элементы строки, в которой он находится. Если таких элементов несколько, обработать каждую строку, содержащую такой элемент.

Общий метод решения

Разреженная матрица – это матрица с преимущественно нулевыми элементами. Если ненулевых элементов матрицы больше, чем нулевых, то матрица считается плотной.

Сначала считаем количество строк и столбцов, затем считаем матрицу заданных размеров. В каждой строке найдём максимальный элемент и разделим на него все элементы данной строки.

Общие сведения о программе

Аппаратное обеспечение: ноутбук Lenovo Z570
ОС: Linux MINT Tessa
Язык и система программирования: GNU C
Стандарт языка: C99
Число строк программы: 257
Компиляция программы в терминале: make
Вызов программы: ./kp7
Makefile:

```
CC=gcc

kursach7:

	$(CC) -o kp7 main7.c vector.c -lm

clean:

	rm -rf *.o kp7
```

Функциональное назначение

Задача программы состоит в обработке прямоугольной разреженной матрицы с элементами комплексного типа, которая:

1. вводит матрицы различного размера

2. печатает введенные матрицы во внутреннем представлении согласно заданной схеме размещения (цепочка ненулевых элементов в векторе A со строчным индексированием)

3. выполняет преобразования (определяет максимальный по модулю элемент матрицы и разделяет на него все элементы строки, в которой он находится)

4. печатает результат преобразования

Описание программы

1. Подключаем библиотеки **stdio.h, stdlib.h**

2. Определим структуру и размерность матрицы

3. Заполним матрицу

4. Определяем функцию нахождения максимальных элементов в каждой строке и деления на него всех элементов

5. Выведем полученную матрицу

Описание функций программы

Функция/выражение	Описание
printSourceMatrix	Выводит введенную матрицу без изменений
printNewMatrix	Выводит полученную матрицу
printInnerMatrix	Выводит представления (мнимое и действительную)

Входные значения

Вводятся непосредственно в программу

Протокол

```
(py37) ➜  kursach7 cat main7.c
#include <stdio.h>
#include <math.h>
#include "vector.h"

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

double complexModule(const Comp c);
Comp complexDivide(const Comp c1, const Comp c2);
void printSourceMatrix(const int *rows, const Vector *cols, const int m, const int n);
void printInnerMatrix(const int *rows, const Vector *cols, const int m);

int main(void)
{
   const int N = 100;
   int m, n, i, j, isEmptyRow, isMaxInRow, row[N];
   Comp maxComp, tmpComp;
   Vector col;
   Item tmpItem;

   vectorCreate(&col, 0);

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

   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;
   }
```

```
for (i = 0; i < m; i++)
{
   isEmptyRow = 1;

   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 (complexModule(tmpComp) == 0.0)
         continue;

      isEmptyRow = 0;

      if (row[i] == END)
         row[i] = vectorSize(&col);

      tmpItem.ind = j;

      vectorPushBack(&col, tmpItem);

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

      vectorPushBack(&col, tmpItem);
   }

   if (isEmptyRow)
      row[i] = EMPTY;
   else
   {
      tmpItem.ind = EMPTY;

      vectorPushBack(&col, tmpItem);
   }
}

tmpItem.ind = END;

vectorPushBack(&col, tmpItem);

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

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

for (i = 0; i < m; i++)
```

```
    {
        if (row[i] == EMPTY)
            continue;

        for (j = row[i]; j < vectorLoad(&col, j).ind != END && vectorLoad(&col, j).ind != EMPTY;
j++)
        {
            if (vectorLoad(&col, j).ind != COMP)
                continue;

            if (complexModule(vectorLoad(&col, j).c) > complexModule(maxComp))
                maxComp = vectorLoad(&col, j).c;
        }
    }

    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;
    }

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

        if (row[i] == EMPTY)
            continue;

        for (j = row[i]; j < vectorLoad(&col, j).ind != END && vectorLoad(&col, j).ind != EMPTY;
j++)
        {
            if (vectorLoad(&col, j).ind != COMP)
                continue;

            if (complexModule(vectorLoad(&col, j).c) == complexModule(maxComp))
            {
                isMaxInRow = 1;

                break;
            }
        }

        if (!isMaxInRow)
            continue;

        for (j = row[i]; j < vectorLoad(&col, j).ind != END && vectorLoad(&col, j).ind != EMPTY;
j++)
```

```
        {
            if (vectorLoad(&col, j).ind == COMP)
            {
                tmpItem = vectorLoad(&col, j);
                tmpItem.c = complexDivide(vectorLoad(&col, j).c, maxComp);

                vectorSave(&col, j, tmpItem);
            }
            else if (vectorLoad(&col, j).ind != EMPTY)
                continue;
        }
    }

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

    vectorDestroy(&col);

    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;
}

void printSourceMatrix(const int *rows, const Vector *cols, const int m, const int n)
{
    int i, j, k;

    for (i = 0; i < m; i++)
    {
        if (rows[i] == EMPTY)
        {
            for (j = 0; j < n; j++)
                printf("(%.2lf, %.2lf) ", 0.0, 0.0);

            printf("\n");
```

```
            continue;
        }

        k = 0;
        j = rows[i];

        while (k < n)
        {
            if (vectorLoad(cols, j).ind == EMPTY)
            {
                printf("(%.2lf, %.2lf) ", 0.0, 0.0);

                k++;

                continue;
            }

            while (k < vectorLoad(cols, j).ind)
            {
                printf("(%.2lf, %.2lf) ", 0.0, 0.0);

                k++;
            }

            printf("(%.2lf, %.2lf) ", vectorLoad(cols, j + 1).c.a, vectorLoad(cols, j + 1).c.b);

            j += 2;
            k++;
        }

        printf("\n");
    }
}

void printInnerMatrix(const int *rows, const Vector *cols, const int m)
{
    int i, j;

    printf("Массив M:\n");

    for (i = 0; i < m; i++)
        printf("%d ", rows[i]);

    printf("\nМассив A:\n");

    if (vectorLoad(cols, 0).ind == END)
    {
        printf("Пуст\n");

        return;
```

```
    }

    for (i = 0; vectorLoad(cols, i).ind != END; i++)
        if (vectorLoad(cols, i).ind == COMP)
            printf("(%.2lf, %.2lf) ", vectorLoad(cols, i).c.a, vectorLoad(cols, i).c.b);
        else
            printf("%d ", vectorLoad(cols, i).ind);

    printf("\n");
}%                                                    (py37) ➜  kursach7 cat 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;
}%                                                    (py37) ➜  kursach7 cat 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
(py37) ➜  kursach7 make
gcc -o kp7 main7.c vector.c -lm
```

Заключение

Мы научились работе с разреженными матрицами в языке Си и получили дополнительный опыт в работе с файлами и написанием инструкций для компилятора. В целом после выполнения данной работы я нашёл для себя новые способы работы с файлами с большей эффективностью, поэтому считаю выполненную работу полезной для становления программистом.