МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт

(национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра 806: «Вычислительная математика и программирование»

Курсовой проект

по курсу «Вычислительные системы»

I I семестр

Задание 7. «Разреженные матрицы.»

|  |  |
| --- | --- |
| Группа: | М8о-107б-18 |
| Студент: | Тояков Артем |
| Преподаватель: | Ридли Александра Николаевна |
| Оценка: |  |
| Дата: |  |

**Оглавление**

[1.Постановка задачи 3](#_Toc12253662)

[2.Структура программы 3](#_Toc12253663)

[3.Описание программы 4](#_Toc12253664)

[3.1 Complex.h: 4](#_Toc12253665)

[3.2 Makefile: 4](#_Toc12253666)

[3.3 Main.c: 5](#_Toc12253667)

[3.4 Simetric\_vector: 7](#_Toc12253668)

[3.5 Functions.c: 8](#_Toc12253669)

[4.Результат работы 12](#_Toc12253670)

[5.Вывод 13](#_Toc12253671)

[6.Список литературы 13](#_Toc12253672)

# 1.Постановка задачи

Составить программу на языке СИ с функциями или процедурами для обработки прямоугольных разреженных матриц с элементами комплексных чисел.

Вариант: 3,10.

Схема размещения: 3 вектора.

PI - столбец, YE — значение, CIP — индекс начала n-ой строки;

Задание: Вычислить матрочлен – многочлен первой степени от разреженной матрицы: (a\*M + b\*E), где Е – единичная матрица, a и b –числовые константы.

# 2.Структура программы

В данной работе моя программа состоит из четырех модулей:

1. main.c
2. simetric\_vector.c
3. Makefile
4. function.c
5. complex.h

В function.c содержатся все функции (сложение матриц и т. д.). Complex.h является заголовочным файлом function.c. Также он содержит в себе структуры комплексного числа, вектора и массива значений. Также достаточно простой Makefille и main.c, в котором я ввожу матрицы, печатаю и вызываю соответствующие функции. В simetric\_vector.c содержатся все функции, связанные с реализацией всех динамических структур, использованных в программе.

# 3.Описание программы

3.1 Complex.h:

Как и отмечалось ранее содержит описание структур, а также ссылки на функции из function.c и simetric\_vector.c.

#ifndef COMPLEX\_H\_

#define COMPLEX\_H\_

#include "stdio.h"

#define compl 2

#include "stdbool.h"

#include "stdlib.h"

typedef struct {

double re;

double im;

} complex;

typedef struct {

complex \*YE;

int \*PI;

int \*CIP;

int ind;

int sizeCIP;

int size;

int cips;

} vector;

void check\_count\_of\_columns\_and\_lines(int fl,int fc);// проверяет равно ли количество строк количеству столбцов

void addCIP(vector \*vect,int k);

void addPI(vector \*v,int j);

void sum\_vector(vector \*v, vector \*c,vector \*s); // сумма

double addYE(vector \*v,double c, double b);

void init\_vector(vector \*v);

void destroy\_vector(vector \*v);

void print\_matrix(vector \*v, int columns,int line);

void print\_vector(vector \*v);

void mult(vector \*v, int a);

// за все остальное говорят названия

#endif

3.2 Makefile:

Довольно простой Makefile состоящий из опций make и make clean.

Используются флаги: -g -std=c99 -Wextra -pedantic -Werror

CC = gcc

CFLAGS = -g -std=c99 -Wextra -pedantic -Werror

OBJ = main.o simetric\_vector.o function.o

main: $(OBJ)

$(CC) $(CCFLAGS) -o main $(OBJ)

main.o: main.c complex.h

function.c:complex.h

simetric\_vector.c:complex.h

clean:

@rm -r \*.o main

3.3 Main.c:

В main.c запускаются все необходимые функции. Хотелось бы отметить, что все элементы вводятся с консоли и именно в main.c происходит их добавление в вектор.

#include "stdio.h"

#include "complex.h"

#include "stdlib.h"

#define compl 2

void help() {

printf("Complex number: x + y\*i;y != 0;\n");

printf("At the terminal 3 + 4i = 3 4;\n");

}

int main(int argc, char \*argv[])

{

int r = 1;

vector first;

vector second;

double real\_part;

double imaginary;

int firlin, fircol;

printf("The first matrix\n");

printf("lines ");

scanf("%d", &firlin);

printf("columns ");

scanf("%d", &fircol);

check\_count\_of\_columns\_and\_lines(firlin,fircol);

init\_vector(&first);

help();

for(int i = 0; i < firlin;i++) {

for(int j = 0; j < fircol; j++) {

scanf("%lf", &real\_part);

scanf("%lf", &imaginary);

if (imaginary != 0 || real\_part != 0) {

(&first)->size++;

(&first)->cips++;

addYE(&first,real\_part,imaginary);

if(r == 1) {

addCIP(&first,(&first)->cips);

r = 0;

}

addPI(&first,j);

}

}

if(r == 1) {

addCIP(&first,0);

}

r = 1;

}

print\_vector(&first);

r = 1;

init\_vector(&second);

help();

for(int i = 0; i < firlin;i++) {

(&second)->cips++;

(&second)->size++;

addYE(&second,1,0);

addCIP(&second,r);

addPI(&second,r - 1);

r++;

}

print\_vector(&second);

vector third;

init\_vector(&third);

printf("Your a and b?\n");

int a,b;

scanf("%d", &a);

scanf("%d", &b);

if(a == 0 && b == 0) {

printf("Result matrix is zero matrix\n");

destroy\_vector(&first);

destroy\_vector(&second);

destroy\_vector(&third);

return 0;

} else if (a == 0) {

mult(&second,b);

print\_matrix(&second,fircol,firlin);

destroy\_vector(&first);

destroy\_vector(&second);

destroy\_vector(&third);

return 0;

} else if (b == 0) {

mult(&first,a);

print\_matrix(&first,fircol,firlin);

destroy\_vector(&first);

destroy\_vector(&second);

destroy\_vector(&third);

return 0;

}

mult(&second,b);

mult(&first,a);

sum\_vector(&first,&second,&third);

printf("Result matrix\n");

print\_vector(&third);

print\_matrix(&third,fircol,firlin);

destroy\_vector(&first);

destroy\_vector(&second);

destroy\_vector(&third);

return 0;

}

3.4 Simetric\_vector:

В данном разделе описывается вектор. Присутствуют такие функции, как: инициализация, печать, разрушение, заполнение вектора. Также присутствует функция инициализации, проверки на пустоту, разрушение.

#include "complex.h"

void print\_vector(vector \*v) {

printf("CIP ");

for(int i = 0; i < v->sizeCIP;i++) {

printf("%d; ", v->CIP[i]);

}

printf("\n");

if (v->size >= 1) {

printf("YE ");

}

for(int i = 0; i < v->size;i++) {

printf("%.1lf %.1lf; ", v->YE[i].re, v->YE[i].im);

}

if (v->size >= 1) {

printf("\n");

printf("PI ");

}

for(int i = 0; i < v->size;i++) {

printf("%d; ", v->PI[i]);

}

if (v->size >= 1) {

printf("\n");

}

}

void init\_vector(vector \*v) {

v->size = 0;

v->sizeCIP = 0;

v->ind = -1;

v->CIP = (int \*)malloc(sizeof(int));

if (v->CIP == NULL) {

exit(1);

}

v->PI = (int \*)malloc(sizeof(int));

if (v->PI == NULL) {

exit(1);

}

v->YE = (complex \*)malloc(sizeof(complex));

if (v->YE == NULL) {

exit(1);

}

}

void addCIP(vector \*v,int k) {

v->sizeCIP++;

v->CIP = (int \*)realloc(v->CIP, sizeof(int) \* (v->sizeCIP));

v->CIP[v->sizeCIP - 1] = k;

}

void addPI(vector \*v,int j) {

v->PI = (int \*)realloc(v->PI, sizeof(int) \* (v->size));

v->PI[v->size - 1] = j;

}

double addYE(vector \*v,double c, double d) {

v->YE = (complex \*)realloc(v->YE, sizeof(complex) \* (compl \* (v->size)));

(v->YE[v->size - 1]).re = c;

(v->YE[v->size - 1]).im = d;

return 0;

}

void destroy\_vector(vector \*v) {

free(v->PI);

free(v->CIP);

free(v->YE);

}

3.5 Functions.c:

В данном разделе описаны все основные функции. В функции void sum\_vector (vector \*v, vector \*c, vector \*s) алгоритм таков: подбирается нужная ситуация и запускается функция заполнения матрицы. Всего их две: 1) void sum\_of\_one\_vector(vector \*c, vector \*s, int ogr, int a) и 2) void sum\_of\_two\_vectors (vector \*v, vector \*c, vector \*s, int ogr\_f, int ogr\_s, int a). Первая функция запускается в случае, когда строка одной матрицы пустая, а строка второй матрицы не пустая. Вторая функция запускается, когда обе строки не пустые. Также присутствует печать матрицы: в функции анализируется подходящая ситуация. Если строка нулевая то печатаются нули, иначе запускается функция void print\_line (vector \*v, int columns, int line, int limit).

#include "complex.h"

void check\_count\_of\_columns\_and\_lines(int fl,int fc) {

if (fl != fc || fl == 0 || fc == 0) {

printf("ERROR\n");

exit(2);

}

}

double sum\_of\_re(vector \*v, vector \*c, int a,int b) {

return (v->YE[a].re + c->YE[b].im);

}

double sum\_of\_im(vector \*v,vector \*c, int a,int b) {

return (v->YE[a].im + c->YE[b].im);

}

void mult(vector \*v, int a) {

for(int i = 0; i < v->size; i++) {

v->YE[i].re = a \* v->YE[i].re;

v->YE[i].im = a \* v->YE[i].im;

}

}

int v\_cip(vector \*v,int c) {

c++;

while((v->CIP[c] == 0) && (c < v->sizeCIP)) {

c++;

}

if(v->CIP[c] != 0) {

return v->CIP[c];

} else {

return v->ind;

}

}

// сложение когда обе строки не пустые

void sum\_of\_two\_vectors(vector \*v,vector \*c,vector \*s,int ogr\_f,int ogr\_s,int a) {

int nv = v->CIP[a] - 1;

int mc = c->CIP[a] - 1;

int r = 1;

while((nv < ogr\_f) && (mc < ogr\_s)) {

if(v->PI[nv] > c->PI[mc]) {

s->size++;

addPI(s,c->PI[mc]);

addYE(s,c->YE[mc].re,c->YE[mc].im);

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

mc++;

} else if(v->PI[nv] < c->PI[mc]) {

s->size++;

addPI(s,v->PI[nv]);

addYE(s,v->YE[nv].re,v->YE[nv].im);

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

nv++;

} else if(v->PI[nv] == c->PI[mc]) {

if((sum\_of\_re(v,c,nv,mc) != 0) || (sum\_of\_im(v,c,nv,mc) != 0)) {

s->size++;

addPI(s,v->PI[nv]);

addYE(s,sum\_of\_re(v,c,nv,mc),sum\_of\_im(v,c,nv,mc));

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

}

mc++;

nv++;

}

}

if((nv == ogr\_f) && (mc < ogr\_s)) {

for (int b = mc; b < ogr\_s;b++) {

s->size++;

addYE(s,c->YE[b].re,c->YE[b].im);

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

addPI(s,c->PI[b]);

}

} else if((mc == ogr\_s) && (nv < ogr\_f)) {

for(int b = nv;b < ogr\_f;b++) {

s->size++;

addYE(s,v->YE[b].re,v->YE[b].im);

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

addPI(s,v->PI[b]);

}

}

if(r == 1) {

addCIP(s,0);

}

}

void print\_line(vector \*v,int columns,int line,int limit) {

int amount = 0;

int b = v->CIP[line] - 1;

while(amount < columns) {

if((amount < v->PI[b]) && (b < limit)) {

amount++;

printf("0 + 0i ");

} else if((amount > v->PI[b]) && (b < limit)){

b++;

printf("0 ");

} else if((amount == v->PI[b]) && (b < limit)) {

printf("%.0lf + %.0lfi ", v->YE[b].re, v->YE[b].im);

amount++;

b++;

} else {

amount++;

printf("0 + 0i ");

}

}

}

void print\_matrix(vector \*v, int columns,int line) {

int a = 0;

if(v->sizeCIP != 1) {

while(a < v->sizeCIP - 1) {

if(v->CIP[a] == 0) {

for(int i = 0;i < columns;i++) {

printf("0 + 0i ");

}

} else if(v->CIP[a] != 0) {

if(v->CIP[a + 1] - v->CIP[a] > 0) {

print\_line(v,columns,a,v->CIP[a + 1] - 1);

} else if(v->CIP[a + 1] - v->CIP[a] < 0) {

if(v\_cip(v,a) == v->ind) {

print\_line(v,columns,a,v->size);

} else if(v\_cip(v,a) - v->CIP[a] > 0) {

int temp = v\_cip(v,a) - 1;

print\_line(v,columns,a,temp);

}

}

}

a++;

printf("\n");

}

if(v->CIP[a] == 0) {

for(int i = 0;i < columns;i++) {

printf("0 + 0i ");

}

} else {

print\_line(v,columns,a,v->size);

}

printf("\n");

} else {

print\_line(v,columns,a,v->size);

}

}

void sum\_of\_one\_vector(vector \*c,vector \*s,int ogr,int a) {

int r = 1;

for(int b = c->CIP[a] - 1;b < ogr;b++) {

s->size++;

addYE(s,c->YE[b].re,c->YE[b].im);

if(r == 1) {

addCIP(s,s->size);

r = 0;

}

addPI(s,c->PI[b]);

}

}

void sum\_vector(vector \*v, vector \*c,vector \*s) {

int a = 0;

int r = 1;

if(v->sizeCIP != 1) {

while(a < v->sizeCIP - 1) {

if(v->CIP[a] == 0 || c->CIP[a] == 0) {

if(v->CIP[a] == 0 && c->CIP[a] == 0) {//ok

if(r == 1) {

addCIP(s,0);

r = 0;

}

} else if((v->CIP[a] == 0) && (c->CIP[a] != 0)) {//ok

if(c->CIP[a + 1] - c->CIP[a] > 0) {

sum\_of\_one\_vector(c,s,c->CIP[a + 1] - 1,a);

} else if(c->CIP[a + 1] - c->CIP[a] < 0) {

if(v\_cip(c,a) - c->CIP[a] > 0) {

int temp = v\_cip(c,a) - 1;

sum\_of\_one\_vector(c,s,temp,a);

} else if(v\_cip(c,a) == c->ind) {

sum\_of\_one\_vector(c,s,c->size,a);

}

}

} else if((v->CIP[a] != 0) && (c->CIP[a] == 0)) {//ok

if(v->CIP[a + 1] - v->CIP[a] > 0) {

sum\_of\_one\_vector(v,s,v->CIP[a + 1] - 1,a);

} else if(v->CIP[a + 1] - v->CIP[a] < 0) {

if(v\_cip(v,a) - v->CIP[a] > 0) {

int temp = v\_cip(v,a) - 1;

sum\_of\_one\_vector(v,s,temp,a);

} else if(v\_cip(v,a) == v->ind) {

sum\_of\_one\_vector(v,s,v->size,a);

}

}

}

} else {

if((v->CIP[a+1] - v->CIP[a] > 0) && (c->CIP[a+1] - c->CIP[a] > 0)) {//ok2

sum\_of\_two\_vectors(v,c,s,v->CIP[a+1] - 1,c->CIP[a+1] - 1,a);

} else if((v->CIP[a+1] - v->CIP[a] < 0) && (c->CIP[a+1] - c->CIP[a] < 0)) {//ok3

if ((v\_cip(v,a) - v->CIP[a] > 0) && (v\_cip(c,a) - c->CIP[a] > 0)) {

int temp\_v = v\_cip(v,a) - 1;

int temp\_c = v\_cip(c,a) - 1;

sum\_of\_two\_vectors(v,c,s,temp\_v,temp\_c,a);

} else if ((v\_cip(v,a) - v->CIP[a] > 0) && (v\_cip(c,a) == c->ind)) {

int temp = v\_cip(v,a) - 1;

sum\_of\_two\_vectors(v,c,s,temp,c->size,a);

} else if ((v\_cip(v,a) == v->ind) && (v\_cip(c,a) - c->CIP[a] > 0)) {

int temp = v\_cip(c,a) - 1;

sum\_of\_two\_vectors(v,c,s,v->size,temp,a);

} else if((v\_cip(v,a) == v->ind) && (v\_cip(c,a) == c->ind)) {

sum\_of\_two\_vectors(v,c,s,v->size,c->size,a);

}

} else if((v->CIP[a+1] - v->CIP[a] < 0) && (c->CIP[a+1] - c->CIP[a] > 0)) {//ok4

if(v\_cip(v,a) - v->CIP[a] > 0) {

int temp = v\_cip(v,a) - 1;

sum\_of\_two\_vectors(v,c,s,temp,c->CIP[a+1] - 1,a);

} else if(v\_cip(v,a) == v->ind) {

sum\_of\_two\_vectors(v,c,s,v->size,c->CIP[a+1] - 1,a);

}

} else if((v->CIP[a+1] - v->CIP[a] > 0) && (c->CIP[a+1] - c->CIP[a] < 0)) {//ok5

if(v\_cip(c,a) - c->CIP[a] > 0) {

int temp = v\_cip(c,a) - 1;

sum\_of\_two\_vectors(v,c,s,v->CIP[a + 1] - 1,temp,a);

} else if(v\_cip(c,a) == c->ind) {

sum\_of\_two\_vectors(v,c,s,v->CIP[a + 1] - 1,c->size,a);

}

}

}

a++;

r = 1;

}

if ((v->CIP[a] == 0) && (c->CIP[a] == 0)) {//ok

if(r == 1) {

addCIP(s,0);

r = 0;

}

} else if((v->size - (v->CIP[a] - 1) > 0) && (c->CIP[a] == 0)) {//ok

sum\_of\_one\_vector(v,s,v->size,a);

} else if((v->CIP[a] == 0) && (c->size - (c->CIP[a] - 1) > 0)) { //ok

sum\_of\_one\_vector(c,s,c->size,a);

} else if ((v->size - (v->CIP[a]- 1) > 0) && (c->size - (c->CIP[a] - 1) > 0)) {//ok

sum\_of\_two\_vectors(v,c,s,v->size,c->size,a);

}

} else {//ok

sum\_of\_two\_vectors(v,c,s,v->size,c->size,a);

}

}

# 4.Результат работы

➜ kp7 git:(master) ✗ make

gcc -g -std=c99 -Wextra -pedantic -Werror -c -o main.o main.c

gcc -g -std=c99 -Wextra -pedantic -Werror -c -o simetric\_vector.o simetric\_vector.c

gcc -g -std=c99 -Wextra -pedantic -Werror -c -o function.o function.c

gcc -o main main.o simetric\_vector.o function.o

➜ kp7 git:(master) ✗ ls

complex.h function.c function.o main main.c main.o Makefile simetric\_vector.c simetric\_vector.o

➜ kp7 git:(master) ✗ ./main

The first matrix

lines 3

columns 3

Complex number: x + y\*i;y != 0;

At the terminal 3 + 4i = 3 4;

1 2 0 0 0 0

0 0 0 0 1 1

0 0 0 0 0 0

CIP 1; 2; 0;

YE 1.0 2.0; 1.0 1.0;

PI 0; 2;

Complex number: x + y\*i;y != 0;

At the terminal 3 + 4i = 3 4;

CIP 1; 2; 3;

YE 1.0 0.0; 1.0 0.0; 1.0 0.0;

PI 0; 1; 2;

Your a and b?

3 5

Result matrix

CIP 1; 2; 4;

YE 3.0 6.0; 5.0 0.0; 3.0 3.0; 5.0 0.0;

PI 0; 1; 2; 2;

3 + 6i 0 + 0i 0 + 0i

0 + 0i 5 + 0i 3 + 3i

0 + 0i 0 + 0i 5 + 0i

# 5.Вывод

Реализовав комплексную разреженную матрицу, можно сделать вывод, что таким образом с помощью схемы размещения можно экономить память. Представление матриц на векторах оказывается очень удобным, например, когда мы хотим реализовать матрицу структурно.

# 6.Список литературы

1.Гайсарян С.С., Зайцев В.Е. «Курс информатики» Москва, Издательство МАИ 1993