 Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу «Языки и методы программирования»**

**II Семестр**

**Задание 7**

**Разреженные матрицы**

|  |  |
| --- | --- |
| Студент: | Почечура А.А. |
| Группа: | М8О-106Б-20 |
| Преподаватель: | Дубинин А.В. |
|  |  |
| Оценка: |  |
| Дата: |  |
| Подпись: |  |

**Москва, 2021 г.**

**Оглавление**

[1. Введение 3](#_Toc69406695)

[2. Разреженные матрицы 3](#_Toc69406696)

[3. Задание 3](#_Toc69406697)

[4. Общий метод решения 5](#_Toc69406698)

[5. Общее сведения о программе 5](#_Toc69406699)

[6. Логическая структура 5](#_Toc69406700)

[7. Текст программы 5](#_Toc69406701)

[8. Таблица переменных и констант 11](#_Toc69406702)

[9. Входные данные 12](#_Toc69406703)

[10. Выходные данные 12](#_Toc69406704)

[11. Сложностные оценки 12](#_Toc69406705)

[12. Демонстрация работы программы 13](#_Toc69406706)

[13. Заключение 15](#_Toc69406707)

[14. Список используемых источников 15](#_Toc69406708)

# **Введение**

Для некоторых задач, которые вычисляются на компьютерах требуются матрицы, которые необходимо как-то хранить и обрабатывать.

Матрицы можно хранить в памяти компьютера целиком, однако мы знаем, что матрица – сплошная структура и, если предположить, что мы не будем хранить в памяти один из видов элементов (допустим нулевой), помня какое значение у элемента, если он не хранится в памяти, это позволит уменьшить расход памяти для матриц, в которых один из элементов встречается много больше других, ускорить время доступа к остальным элементам и операции над ними.

# **Разреженные матрицы**

Разреженные матрицы – это матрицы, которые в основном содержат нулевые элементы. Преимущество заключается в том, что нули в матричных операциях воспринимаются как пустые компоненты.

Использование цепочки ненулевых элементов со строчным индексированием удобно тем, что имеется доступ ко всем ненулевым элементам исходной матрицы, а также, косвенно, к их положениям в матрице, что упрощает произведение арифметических операций с ними, ведь все остальные элементы матрицы нулевые.

# **Задание**

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.

Вариант схемы размещения матрицы определяется по формуле ((N + 3) % 4) + 1‚ где N - номер студента по списку в группе, Вариант преобразования определяется по формуле ((N - 1) % 11) + 1. Вариант физического представления (1 - отображение на массив. 2 - отображение на динамические структуры) определяются по формуле ([1.5\*((3+M) % 9)] +N) %2+1, где M - номер группы. В случае использования динамических структур индексы заменяются соответствующими ссылками.

Вариант схемы размещения матрицы:



Вариант преобразования:

7. Найти строку, содержащую наибольшее количество ненулевых элементов, и напечатать ее номер и сумму элементов этой строки. Если таких строк несколько, обработать все.

# **Общий метод решения**

Работа идёт с тремя программами языка Си – основной и двумя вспомогательными для работы с двумя типами динамических массивов и матрицей, двумя заголовочными файлами (содержат описания структур, необходимых для работы соответствующих программ).

Основная программа считывает матрицу и её характеристики, выводит её внутреннее и естественные представления, находит строки, в которых максимальное количество ненулевых элементов и выводит их номера и сумму элементов этой строки.

# **Общее сведения о программе**

Операционная система семейства Unix, наименование Ubuntu, версия 20.04. Язык программирования – Си.

# **Логическая структура**

Подаём на вход количество строк в матрице, количество столбцов и саму матрицу. Помещаем данные в структуру matrix, разбивая на три массива: элементов, номеров столбцов этих элементов и количества элементов по строкам. Выводим внутреннее и естественное представления матрицы. Далее осуществляем проход по массиву с количеством элементов по строкам и находим строки, в которых наибольшее количество ненулевых элементов. Это количество записываем в переменную. Осуществляем второй проход по этому же массиву, чтобы вывести номера строк и сумму их элементов.

# **Текст программы**

**matrix.с**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include "vector\_dbl.h"

#include "vector\_ll.h"

typedef struct{

long long m;

long long n;

vector\_ll cip;

vector\_ll pi;

vector\_double ye;

} matrix;

void create\_matrix(matrix \*matr){

matr->m=0;

matr->n=0;

vll\_create(&matr->cip);

vll\_create(&matr->pi);

vd\_create(&matr->ye);

}

void read\_matrix(matrix \*matr){

scanf("%lli%lli",&matr->m,&matr->n);

double a;

long long last=0;

vll\_push(&matr->cip,0);

for(long long i=0;i<matr->m;i++){

for(long long j=0;j<matr->n;j++){

scanf("%lf",&a);

if(a!=0){

vll\_push(&matr->pi,j);

vd\_push(&matr->ye,a);

}

last=vll\_size(&matr->pi);

}

vll\_push(&matr->cip,last);

}

printf("\n");

}

void print\_cip\_ip\_ye\_matrix(matrix \*matr){

printf("CIP: ");

for(long long i=0;i<vll\_size(&matr->cip);i++){

printf("%lli ",vll\_get(&matr->cip,i));

}

printf("\nPI: ");

for(long long i=0;i<vll\_size(&matr->pi);i++){

printf("%lli ",vll\_get(&matr->pi,i));

}

printf("\nYE: ");

for(long long i=0;i<vd\_size(&matr->ye);i++){

printf("%f ",vd\_get(&matr->ye,i));

}

printf("\n\n");

}

void printf\_matrix(matrix \*matr){

long long count=0;

for(long long i=0;i<matr->m;i++){

if(vll\_get(&matr->cip,i)==vll\_get(&matr->cip,i+1)){

for(long long i=0;i<matr->n;i++){

printf("0 ");

}

} else {

for(long long j=vll\_get(&matr->cip,i);j<vll\_get(&matr->cip,i+1);j++){

if(vll\_get(&matr->pi,j)==count){

printf("%f ",vd\_get(&matr->ye,j));

} else {

j--;

printf("0 ");

}

count++;

}

while(count<matr->n){

printf("0 ");

count++;

}

count=0;

}

printf("\n");

}

printf("\n");

}

void task(matrix \*matr){

long long max=0;

for(long long i=0;i<matr->m;i++){

if(vll\_get(&matr->cip,i+1)-vll\_get(&matr->cip,i)>max){

max=vll\_get(&matr->cip,i+1)-vll\_get(&matr->cip,i);

}

}

if(max==0){

printf("All elements of matrix are equal to zero\n\n");

} else{

long long sum=0;

for(long long i=0;i<=matr->m;i++){

if(vll\_get(&matr->cip,i+1)-vll\_get(&matr->cip,i)==max){

printf("Number of matrix line: %lli\n",i+1);

for(int j=vll\_get(&matr->cip,i);j<vll\_get(&matr->cip,i+1);j++){

sum=sum+vd\_get(&matr->ye,j);

}

printf("Sum of elements of matrix line: %lli\n\n",sum);

sum=0;

}

}

}

}

void destroy\_matrix(matrix \*matr){

matr->m=0;

matr->n=0;

vll\_destroy(&matr->cip);

vll\_destroy(&matr->pi);

vd\_destroy(&matr->ye);

}

int main(){

matrix matr;

create\_matrix(&matr);

read\_matrix(&matr);

print\_cip\_ip\_ye\_matrix(&matr);

printf\_matrix(&matr);

task(&matr);

destroy\_matrix(&matr);

}

**vector\_dbl.h**

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

double \*buffer;

long long size;

long long cap;

long long first;

} vector\_double;

void vd\_create(vector\_double \*v);

void vd\_destroy(vector\_double \*v);

long long vd\_size(vector\_double \*v);

bool vd\_grow\_buffer(vector\_double \*v);

bool vd\_push\_back(vector\_double \*v, double f);

double vd\_get(vector\_double \*v, long long i);

**vector\_dbl.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#include "vector.h"

void vd\_create(vector\_double \*v){

v->size=0;

v->first=0;

v->cap=0;

v->buffer=NULL;

}

void vd\_destroy(vector\_double \*v){

v->size=0;

v->first=0;

v->cap=0;

free(v->buffer);

v->buffer=NULL;

}

double vd\_get(vector\_double \*v,long long i){

return v->buffer[(v->first+i)%v->cap];

}

long long vd\_size(vector\_double \*v){

return v->size;

}

bool vd\_grow\_buffer(vector\_double \*v){

long long tmp=v->cap\*3/2;

if(v->cap==0){

tmp=9;

}

double \*n=realloc(v->buffer,sizeof(double)\*tmp);

if(n==NULL){

return false;

} else {

v->buffer=n;

for(long long i=v->cap-v->first-1;i>=0;i--){

v->buffer[(tmp+i-(v->cap-v->first))%tmp]=v->buffer[(v->first+i)%tmp];

v->buffer[(v->first+i)%tmp]=0;

}

v->first=(tmp-(v->cap-v->first))%tmp;

v->cap=tmp;

return true;

}

}

bool vd\_push\_back(vector\_double \*v,double val){

if(v->size>=v->cap){

if(!(vd\_grow\_buffer(v))){

return false;

}

}

v->buffer[(v->first+v->size)%v->cap]=val;

v->size++;

return true;

}

**vector\_ll.h**

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

long long \*buffer;

long long size;

long long cap;

long long first;

} vector\_ll;

void vll\_create(vector\_ll \*v);

void vll\_destroy(vector\_ll \*v);

long long vll\_size(vector\_ll \*v);

bool vll\_grow\_buffer(vector\_ll \*v);

bool vll\_push\_back(vector\_ll \*v, long long f);

long long vll\_get(vector\_ll \*v, long long i);

**vector\_ll.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#include "vector\_ll.h"

void vll\_create(vector\_ll \*v){

v->size=0;

v->first=0;

v->cap=0;

v->buffer=NULL;

}

void vll\_destroy(vector\_ll \*v){

v->size=0;

v->first=0;

v->cap=0;

free(v->buffer);

v->buffer=NULL;

}

long long vll\_get(vector\_ll \*v,long long i){

return v->buffer[(v->first+i)%v->cap];

}

long long vll\_size(vector\_ll \*v){

return v->size;

}

bool vll\_grow\_buffer(vector\_ll \*v){

long long tmp=v->cap\*3/2;

if(v->cap==0){

tmp=9;

}

long long \*n=realloc(v->buffer,sizeof(long long)\*tmp);

if(n==NULL){

return false;

} else {

v->buffer=n;

for(long long i=v->cap-v->first-1;i>=0;i--){

v->buffer[(tmp+i-(v->cap-v->first))%tmp]=v->buffer[(v->first+i)%tmp];

v->buffer[(v->first+i)%tmp]=0;

}

v->first=(tmp-(v->cap-v->first))%tmp;

v->cap=tmp;

return true;

}

}

bool vll\_push\_back(vector\_ll \*v,long long val){

if(v->size>=v->cap){

if(!(vll\_grow\_buffer(v))){

return false;

}

}

v->buffer[(v->first+v->size)%v->cap]=val;

v->size++;

return true;

}

# **Таблица переменных и констант**

Описание переменных и структур в заголовочном файле “vector\_ll.h”:

Структура **vector\_ll:**

|  |  |
| --- | --- |
| long long buffer\* | Указатель на буфер массива |
| long long size; | Количество элементов массива |
| long long cap; | Размер буфера массива |
| long long first; | Номер первого элемента в массиве |

Описание переменных и структур в заголовочном файле “vector\_dbl.h”:

Структура **vector\_double:**

|  |  |
| --- | --- |
| double buffer\* | Указатель на буфер массива |
| long long size; | Количество элементов массива |
| long long cap; | Размер буфера массива |
| long long first; | Номер первого элемента в массиве |

Описание переменных и структур в заголовочном файле “matrix.c”:

Структура **matrix**:

|  |  |
| --- | --- |
| long long m; | Количество строк в матрице |
| long long n; | Количество столбцов в матрице |
| vector\_ll cip | Массив с индексами начала i-той строки в массивах PI и YE |
| vector\_ll pi | Номер столбца значения |
| vector\_dbl ye | Значение |

# **Входные данные**

При запуске программа получает матрицу и её характеристики: число строк и столбцов.

# **Выходные данные**

Входная матрица во внутреннем представлении, входная матрица естественном виде, номера строк с максимальным количеством ненулевых элементов и сумма их элементов.

# **Сложностные оценки**

Пусть в исходной матрице m строк, n столбцов и k ненулевых элементов.

**Создание матрицы**

Временная оценка: O(1)

Пространственная оценка: O(1)

**Считывание матрицы**

Временная оценка: O(n\*m)

Пространственная оценка: O(m)

**Вывод матрицы во внутреннем представлении**

Временная оценка: O(m+k)

Пространственная оценка: O(1)

**Вывод матрицы в естественном представлении**

Временная оценка: O(n\*m)

Пространственная оценка: O(1)

**Функция выполнения задания**

Временная оценка: O(m+k)

Пространственная оценка: O(m+k)

**Удаление матрицы**

Временная оценка: O(1)

Пространственная оценка: O(1)

# **Демонстрация работы программы**

root@DESKTOP-5HM2HTK:~# gcc -std=c99 -pedantic -Wall kursach2.c vector\_functions.c vector\_ll\_functions.c

root@DESKTOP-5HM2HTK:~# ./a.out

12 5

0 0 0 0 0

0 7 0 3 5

1 0 2 0 0

0 0 0 0 0

10 12 13 14 0

0 3 4 5 6

2 0 0 0 0

0 2 0 0 0

1 2 3 4 0

0 2 3 4 0

4 0 1 2 0

0 1 0 0 0

CIP: 0 0 3 5 5 9 13 14 15 19 22 25 26

PI: 1 3 4 0 2 0 1 2 3 1 2 3 4 0 1 0 1 2 3 1 2 3 0 2 3 1

YE: 7.000000 3.000000 5.000000 1.000000 2.000000 10.000000 12.000000 13.000000 14.000000 3.000000 4.000000 5.000000 6.000000 2.000000 2.000000 1.000000 2.000000 3.000000 4.000000 2.000000 3.000000 4.000000 4.000000 1.000000 2.000000 1.000000

0 0 0 0 0

0 7.000000 0 3.000000 5.000000

1.000000 0 2.000000 0 0

0 0 0 0 0

10.000000 12.000000 13.000000 14.000000 0

0 3.000000 4.000000 5.000000 6.000000

2.000000 0 0 0 0

0 2.000000 0 0 0

1.000000 2.000000 3.000000 4.000000 0

0 2.000000 3.000000 4.000000 0

4.000000 0 1.000000 2.000000 0

0 1.000000 0 0 0

Number of matrix line: 5

Sum of elements of matrix line: 49

Number of matrix line: 6

Sum of elements of matrix line: 18

Number of matrix line: 9

Sum of elements of matrix line: 10

root@DESKTOP-5HM2HTK:~# ./a.out

5 3

0 0 0

9 0 8

0 0 0

0 0 0

7 6 0

CIP: 0 0 2 2 2 4

PI: 0 2 0 1

YE: 9.000000 8.000000 7.000000 6.000000

0 0 0

9.000000 0 8.000000

0 0 0

0 0 0

7.000000 6.000000 0

Number of matrix line: 2

Sum of elements of matrix line: 17

Number of matrix line: 5

Sum of elements of matrix line: 13

root@DESKTOP-5HM2HTK:~#

# **Заключение**

Использованные приёмы позволяют хранить огромные разреженные матрицы со сравнительно небольшим заполнением, а также достаточно быстро получать доступ к элементу. Естественно, эффективность использования таких матриц обратно пропорционально зависит от заполненности, чем меньше ненулевых элементов, тем лучше.

# **Список используемых источников**

Методические указания к выполнению курсовых работ. Зайцев В. Е.