**Московский Авиационный Институт**

(Национальный Исследовательский Университет)

Кафедра вычислительной математики и программирования

**Курсовая работа**

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

**II Семестр**

**Задание 7**

**Разреженные матрицы**

Студент: Кондратьев Егор Алексеевич

Группа: М8О-106Б-19

Преподаватель: Дубинин Алексей Владимирович

Москва, 2020

Оглавление

Оглавление

[Оглавление 2](#_Toc44008105)

[Задание 3](#_Toc44008106)

[Общий метод решения 3](#_Toc44008107)

[Функциональное назначение 4](#_Toc44008108)

[Текст программы 4](#_Toc44008109)

[Описание переменных 8](#_Toc44008110)

[Сложность 9](#_Toc44008111)

[Заключение 9](#_Toc44008112)

[Список используемых источников 9](#_Toc44008113)

# 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:

Два вектора.

Вариант преобразования:

5. Умножить вектор-строку на разреженных матрицу и вычислить количество ненулевых элементов результата.

Вариант физического представления:

2. Отображение на массив.

# Общий метод решения

Разреженными называются матрицы, в которых ненулевых элементов много меньше

общего числа элементов.

Хранение разреженной матрицы в памяти должно обеспечивать:

1) экономию памяти

2) быстрый доступ к нулевым и ненулевым элементам по их индексу.

Использование цепочки ненулевых элементов со строчным индексированием удобно тем, что мы имеем доступ ко всем ненулевым элементам исходной матрицы, а также их индексам, и можем производить с ними любые действия, пологая что остальные элементы матрицы нулевые.

# Функциональное назначение

Программа предназначена для демонстрации использования метода хранения разреженной матрицы с помощью использования цепочки ненулевых элементов со строчным индексированием.

# Текст программы

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct

{

    double \*buf;

    size\_t size;

    size\_t capacity;

} vector;

typedef struct

{

    int m;

    int n;

    vector k;

    vector v;

} matrix;

bool v\_init(vector \*v)

{

    double \*newbuf = malloc(64 \* sizeof(double));

    if (newbuf != NULL)

    {

        v->buf = newbuf;

        v->size = 0;

        v->capacity = 64;

        return true;

    }

    return false;

}

void v\_destroy(vector \*v)

{

    free(v->buf);

    v->buf = NULL;

    v->size = 0;

    v->capacity = 0;

}

double v\_get(vector \*v, size\_t i)

{

    return v->buf[i];

}

void v\_set(vector \*v, size\_t i, double val)

{

    v->buf[i] = val;

}

size\_t v\_get\_size(vector \*v)

{

    return v->size;

}

bool v\_set\_size(vector \*v, size\_t new\_size)

{

    if (new\_size > v->capacity)

    {

        size\_t new\_cap = v->capacity \* 3 / 2;

        if (new\_cap < new\_size)

        {

            new\_cap = new\_size;

        }

        if (new\_cap < 64)

        {

            new\_cap = 64;

        }

        double \*new\_buf = realloc(v->buf, new\_cap \* sizeof(double));

        if (new\_buf == NULL)

        {

            return false;

        }

        v->buf = new\_buf;

        v->capacity = new\_cap;

    }

    else if (new\_size \* 3 / 2 < v->capacity)

    {

        size\_t new\_cap = new\_size \* 3 / 2;

        if (new\_cap < 64)

        {

            new\_cap = 64;

        }

        v->buf = realloc(v->buf, new\_cap \* sizeof(double));

        v->capacity = new\_cap;

    }

    for (size\_t i = v->size; i < new\_size; i++)

    {

        v\_set(v, i, 0);

    }

    v->size = new\_size;

    return true;

}

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

{

    if (v\_set\_size(v, v\_get\_size(v) + 1))

    {

        v\_set(v, v\_get\_size(v) - 1, val);

        return true;

    }

    return false;

}

void mat\_init(matrix \*m)

{

    v\_init(&m->k);

    v\_init(&m->v);

}

void mat\_destroy(matrix \*m)

{

    v\_init(&m->k);

    v\_init(&m->v);

    m->m = 0;

    m->n = 0;

}

void mat\_print\_vec(matrix \*mat)

{

    printf("indexes: ");

    for (int i = 0; i < v\_get\_size(&mat->k); i++)

    {

        printf("%d ", (int)v\_get(&mat->k, i));

    }

    printf("\n");

    printf("values: ");

    for (int i = 0; i < v\_get\_size(&mat->v); i++)

    {

        printf("%d ", (int)v\_get(&mat->v, i));

    }

    printf("\n");

}

vector multiplication(vector \*a, matrix \*b)

{

    int l = b->n;

    vector result;

    v\_init(&result);

    int aa = b->m \* l;

    for (int i = 0; i < l; ++i)

        v\_push\_back(&result, 0);

    for (int i = 0; i < l; i++)

    {

        int res\_tmp = 0;

        int f = 0;

        for (int j = i; j < aa; j += l)

        {

            int inde\_b = (int)v\_get(&b->k, j);

            double val\_b = v\_get(&b->v, j);

            if (inde\_b == j)

            {

                res\_tmp += v\_get(a, f) \* val\_b;

            }

            f++;

        }

        v\_set(&result, i, res\_tmp);

    }

    return result;

}

int nonzero\_count(vector result)

{

    int count = 0;

    for (int i = 0; i < v\_get\_size(&result); ++i)

    {

        printf("%f ", v\_get(&result, i));

        if (v\_get(&result, i) != 0)

            count++;

    }

    return count;

}

int main()

{

    vector a;

    matrix b;

    v\_init(&a);

    mat\_init(&b);

    printf("Введите количество строк и количество столбцов матрицы:\n");

    int n, m;

    scanf("%d%d", &m, &n);

    b.m = m;

    b.n = n;

    printf("Введите вектор-строку:\n");

    for (int i = 0; i < n; i++)

    {

    int tmp;

        scanf("%d", &tmp);

        v\_push\_back(&a, tmp);

    }

    printf("Введите матрицу:\n");

    for (int i = 0; i < b.m; i++)

    {

        for (int j = 0; j < b.n; j++)

        {

            int tmp;

            scanf("%d", &tmp);

            if (tmp != 0)

            {

                v\_push\_back(&b.k, i \* b.m + j);

                v\_push\_back(&b.v, (int)tmp);

            }

        }

    }

    mat\_print\_vec(&b);

    vector res;

    v\_init(&res);

    res = multiplication(&a, &b);

    printf("Результирующий вектор:\n");

    int aa = nonzero\_count(res);

    printf("\nКоличество ненулевых элементов результата: %d\n", aa);

    v\_destroy(&a);

    mat\_destroy(&b);

    v\_destroy(&res);

    return 0;

}

# Описание переменных

|  |  |  |
| --- | --- | --- |
| Тип | Название | Описание |
| int | n | Число строк |
| int | m | Число столбцов |
| vector | k | Счётчик прохода по строкам |
| vector | v | Счётчик прохода по столбцам |
| int | inde | Число, ставящееся в соответствие индексам элемента, вычисляется по формуле i\*n+j |
| int | val | Значение элемента по индексу inde |

# Сложность

Умножение – O(k) кол-во ненулевых элементов

Вычислить количество ненулевых элементов результата – O(m) кол-во строк

# 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.