# Отчет по лабораторной работе № VII по курсу
# Алгоритмы и структуры данных

Студент группы М8О-103Б-22 Клименко Виталий Максимович, № по списку 11

Контакты www, e-mail, icq, skype vitalikklimenko96@gmail.com

Работа выполнена: 21 мая 2023 г.

Преподаватель: доцент Никулин С.П.

Входной контроль знаний с оценкой _____

Отчет сдан «  » _____ 202 _ г., итоговая оценка ___

Подпись преподавателя _____

1. **Тема:** Разреженные матрицы

2. **Цель работы:** Составить программу на Си с функциями для обработки прямоугольных разряженных матриц с элементами вещественного типа

3. **Задание** (*вариант № 3, 11, 1*): Транспонировать матрицу относительно побочной диагонали. Проверить является ли она косо-симметрической

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____ с ОП _____ Мб,
НМД _____ Мб. Терминал _____ адрес _____. Принтер _____
Другие устройства _____

*Оборудование ПЭВМ студента, если использовалось:*
Процессор Intel 4x 3.5GHz  с ОП 16 ГБ  НМД HDD 200 ГБ  . Монитор Встроенный 1920x1080
Другие устройства Touchpad Synaptics

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства _____, наименование _____ версия _____
интерпретатор команд _____ версия _____
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных _____

*Программное обеспечение ЭВМ студента, если использовалось:*
Операционная система семейства UNIX, наименование Pop!_OS версия 22.04 jammy
интерпретатор команд bash версия 5.1.16
Система программирования _____ версия _____
Редактор текстов _____ версия _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных на домашнем компьютере _____

6. **Идея, метод, алгоритм** решение задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)

В моем варианте разреженная матрица представлена тремя векторами:
Первый указывает на то, с какого индекса начинается итая строка во втором векторе, второй вектор описывает номер столбца в котором находится элемент, параллельно идущий в третьем векторе.
Для удобства в структуре будем хранить количество строк, столбцов, ненулевых элементов и собственно три вектора, описывающие матрицу
Длина первого вектора равняется количеству строк + 1, для удобства обработки. Длина второго и третьего вектора равны количеству ненулевых элементов

7. **Сценарий выполнения работы** (план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты либо соображения по тестированию)

Помимо базовых функций над разреженными матрицами (аллоцирование пустой матрицы, взятие элемента на итой строке и в житом столбце), для выполнения задания составим несколько функций: функция считывания матрицы из файла, функция транспонирования, функция транспонирования относительно побочной диагонали, функция умножения на число.
Функция проверки на то косо-симметрическая ли матрица будет состоять из транспонирования матрицы (храненим в переменной $t$), умножения матрицы на -1 ($m$) и сравнение ($m \stackrel{?}{=} t$).

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*
*Допущен к выполнению работы.* **Подпись преподавателя** _____

**8. Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)

```
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ ls
./  ../  m1.txt*  m2.txt*  m3.txt*  main.c*  makefile*  protocol.txt*  report.pdf*  smatrix.h*
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat main.c
#define SPARSE_MATRIX_IMPLEMENTATION
#include "smatrix.h"


int main() {
    FILE *f = fopen("m1.txt", "r");
    assert(f != NULL);

    SMatrix m = smatrix_from_file(f);

    printf("Initial matrix:\n");
    printf("(Real matrix)\n");
    smatrix_print(m);
    printf("(Internal structure)\n");
    smatrix_print_debug(m);

    printf(
        "========================================\n"
        "Transposed by secondary diagonal matrix:\n"
    );
    SMatrix t = smatrix_transpose_sec_diag(m);
    printf("(Real matrix)\n");
    smatrix_print(t);
    printf("(Internal structure)\n");
    smatrix_print_debug(t);

    printf("\n");
    if (smatrix_is_skew_symmetric(t)) {
        printf("The matrix is skew-symmetric!\n");
    } else {
        printf("The matrix is not skew-symmetric...\n");
    }

    smatrix_free(&m);
    smatrix_free(&t);
    fclose(f);

    return 0;
}vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat smatrix.h
#ifndef SPARSE_MATRIX_H
#define SPARSE_MATRIX_H

#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>

// ================================
// Structure:
// rows_s = {0, 2, 3}, len(rows_s) <= rows
// cols_i = {
//     0, 1,              (1 2 0)
//     1,               =>(0 3 0)
//     2                  (0 0 4)
// }
// values = {1, 2, 3, 4}

typedef struct SMatrix {
    uint64_t rows;
    uint64_t cols;
    uint64_t elements;
    float *values;
    uint64_t *rows_s;
    uint64_t *cols_i;
} SMatrix;

#define SMATRIX_PRINT_NAMED(m) printf("%s = ", #m); smatrix_print(m)

SMatrix smatrix_alloc(uint64_t rows, uint64_t cols);
SMatrix smatrix_from_file(FILE *f);

SMatrix smatrix_transpose(SMatrix m);
SMatrix smatrix_transpose_sec_diag(SMatrix m);

float smatrix_at(SMatrix m, uint64_t i, uint64_t j);
```

```
void smatrix_print(SMatrix m);
void smatrix_print_debug(SMatrix m);

void smatrix_set(SMatrix *m, uint64_t i, uint64_t j, float v);
void smatrix_mult(SMatrix m, float n);
int smatrix_eq(SMatrix m1, SMatrix m2);

int smatrix_is_skew_symmetric(SMatrix m);

void smatrix_free(SMatrix *m);

#endif // SPARSE_MATRIX_H



#ifdef SPARSE_MATRIX_IMPLEMENTATION
#define SPARSE_MATRIX_IMPLEMENTATION

int char_is_in_float(char cur_c) {
    return ('0' <= cur_c && cur_c <= '9') || cur_c == '.' || cur_c == '-';
}


SMatrix smatrix_alloc(uint64_t rows, uint64_t cols) {
    SMatrix m;
    m.rows = rows;
    m.cols = cols;
    m.rows_s = (uint64_t*) calloc(rows + 1, sizeof(*m.rows_s));
    assert(m.rows_s != NULL);

    m.cols_i = (uint64_t*) calloc(1, sizeof(*m.cols_i));
    assert(m.cols_i != NULL);

    m.values = (float*) calloc(1, sizeof(*m.values));
    assert(m.values != NULL);

    m.elements = 0;

    return m;
}

SMatrix smatrix_from_file(FILE *f) {
    char row_buf[256] = {};
    fgets(row_buf, 256, f);

    uint64_t rows = 0, cols = 0;

    for (uint64_t i = 0; i < strlen(row_buf); ++i) { // pre-count column count
        if (char_is_in_float(row_buf[i]) &&
            (row_buf[i + 1] == ' ')
        ) { // check how many times space or \n is present after float chars
            cols++;
        }
    }
    cols++;

    // printf("cols = %llu\n", cols);

    // TODO: create 3 different vectors like in matrix
    uint64_t *rows_s = (uint64_t*) calloc(2, sizeof(uint64_t));
    assert(rows_s != NULL);

    uint64_t *cols_i = (uint64_t*) calloc(1, sizeof(uint64_t));
    assert(cols_i != NULL);

    float *values = (float*) calloc(1, sizeof(float));
    assert(values != NULL);

    uint64_t elements = 0;

    // float **numbers = (float**) calloc(256, sizeof(float*)); // calloc for zeroing

    do {
        printf("rows = %d, alloced rows_s = %d\n", rows, rows + 2);
        rows_s = (uint64_t*) realloc(rows_s, sizeof(*rows_s) * (rows + 2));
        rows_s[rows + 1] = rows_s[rows];

        uint64_t len = strlen(row_buf), c = 0;
        if (row_buf[len - 1] == '\n') row_buf[len - 1] = '\0';

        for (uint64_t i = 0; i < len; ++i) { // parse current line
            while (row_buf[i] == ' ') ++i; // skip spaces before current number
            if (row_buf[i] == '\n') break;
```

```
        char *num_buf = (char*) calloc(256, sizeof(char)); // current number buffer

        for (uint64_t j = 0; j < len - i; ++j) { // parse current number
            char cur_c = row_buf[i + j];

            if (char_is_in_float(cur_c)) {
                num_buf[j] = cur_c;
            } else {
                i += j; // skip whole number for i
                break;
            }
        }

        if (strlen(num_buf) == 0) {
            free(num_buf);
            break;
        }

        // if (c == 0) { // alloc every time there is a new row
        //     numbers[rows] = (float*) calloc(cols, sizeof(float));
        // }

        // printf("%f at r=%llu, c=%llu\n", strtof(num_buf, NULL), rows, c);
        // numbers[rows][c] = strtof(num_buf, NULL);

        // adding elements to 3 vectors

        c++;

        float number = strtof(num_buf, NULL);
        if (number == 0.f) {
            free(num_buf);
            continue;
        }

        for (uint64_t ind = 0; ind < rows + 2; ++ind) { // literally smatrix_set
            if (rows < ind) {
                // {.., 3, 3, ..} -> {.., 3, 4, ..}
                rows_s[ind]++;
            } else if (rows == ind) {
                elements++;
                // printf("elements = %llu\n", elements);
                // printf("realloc cols_i...\n");
                cols_i = (uint64_t*) realloc(cols_i, elements * sizeof(*cols_i));
                // printf("realloc values...\n");
                values = (float*) realloc(values, elements * sizeof(*values));

                uint64_t lb = rows_s[rows];
                uint64_t rb = elements - 1;

                // printf("lb = %llu, rb = %llu\n", lb, rb);

                for (uint64_t move = rb; move > lb; --move) { // move right column indecies and values to right
                    cols_i[move] = cols_i[move - 1]; // move - 1 to safely get to 0
                    values[move] = values[move - 1];
                }

                // printf("set cols...\n");
                cols_i[lb] = c - 1;
                // printf("set values...\n");
                values[lb] = strtof(num_buf, NULL);
            }

        }

        free(num_buf);
    }

    rows++;
} while (fgets(row_buf, 256, f) != NULL);

// printf("dims: %llu, %llu\n", rows, cols);

SMatrix m = smatrix_alloc(rows, cols);

m.elements = elements;

free(m.rows_s);
free(m.cols_i);
free(m.values);

m.rows_s = rows_s;
m.cols_i = cols_i;
m.values = values;
```

```
    return m;
}


SMatrix smatrix_transpose(SMatrix m) {
    SMatrix t = smatrix_alloc(m.cols, m.rows);

    for (uint64_t i = 0; i < m.rows; ++i) {
        for (uint64_t j = 0; j < m.cols; ++j) {
            float n = smatrix_at(m, i, j);
            if (n != 0.f) {
                smatrix_set(&t, j, i, n);
            }
        }
    }

    return t;
}

SMatrix smatrix_transpose_sec_diag(SMatrix m) {
    SMatrix t = smatrix_alloc(m.cols, m.rows);

    for (uint64_t i = 0; i < m.rows; ++i) {
        for (uint64_t j = 0; j < m.cols; ++j) {
            float n = smatrix_at(m, i, j);
            if (n != 0.f) {
                smatrix_set(&t, m.cols - j - 1, m.rows - i - 1, n);
            }
        }
    }

    return t;
}


float smatrix_at(SMatrix m, uint64_t i, uint64_t j) {
    assert(i < m.rows);
    assert(j < m.cols);

    for (uint64_t c = m.rows_s[i]; c < m.rows_s[i + 1]; ++c) {
        if (m.cols_i[c] == j) return m.values[c];
    }

    return 0;
}


void smatrix_print(SMatrix m) {
    printf("[\n");
    for (uint64_t i = 0; i < m.rows; ++i) {
        for (uint64_t j = 0; j < m.cols; ++j) {
            printf("%8.3f ", smatrix_at(m, i, j));
        }
        printf("\n");
    }
    printf("]\n");
}

void smatrix_print_debug(SMatrix m) {
    printf("rows = %llu, cols = %llu, elements = %llu\n", m.rows, m.cols, m.elements);

    printf("rows_s: [");
    for (uint64_t i = 0; i < m.rows; ++i)
        printf("%llu ", m.rows_s[i]);
    printf("(%llu) ", m.rows_s[m.rows]);
    printf("\b]\n");

    if (m.elements == 0)
        return;

    printf("cols_i: [");
    for (uint64_t i = 0; i < m.elements; ++i)
        printf("%llu ", m.cols_i[i]);
    printf("\b]\n");

    printf("values: [");
    for (uint64_t i = 0; i < m.elements; ++i)
        printf("%f ", m.values[i]);
    printf("\b]\n");
}


void smatrix_set(SMatrix *m, uint64_t i, uint64_t j, float v) {
    assert(i < m->rows);
    assert(j < m->cols);
```

```
    for (uint64_t ind = 0; ind < m->rows + 1; ++ind) {
        if (i < ind) {
            // {.., 3, 3, ..} -> {.., 3, 4, ..}
            // printf("++m.rows_s[%I64d]\n", ind);
            m->rows_s[ind]++;
        } else if (i == ind) {
            m->elements++;
            // printf("m->elements = %llu\n", m->elements);
            // printf("realloc cols_i...\n");
            m->cols_i = (uint64_t*) realloc(m->cols_i, m->elements * sizeof(*m->cols_i));
            // printf("realloc values...\n");
            m->values = (float*) realloc(m->values, m->elements * sizeof(*m->values));

            // printf("lb...\n");
            uint64_t lb = m->rows_s[i];
            // printf("rb...\n");
            uint64_t rb = m->elements - 1;

            // printf("lb = %llu, rb = %llu\n", lb, rb);

            for (uint64_t move = rb; move > lb; --move) { // move right column indecies and values to right
                m->cols_i[move] = m->cols_i[move - 1]; // move - 1 to safely get to 0
                m->values[move] = m->values[move - 1];
            }

            m->cols_i[lb] = j;
            m->values[lb] = v;
        }
    }
}

void smatrix_mult(SMatrix m, float n) {
    if (n == 0.f) {
        m = smatrix_alloc(m.rows, m.cols);
        return;
    }

    for (uint64_t i = 0; i < m.elements; ++i) {
        m.values[i] *= n;
    }
}

int smatrix_eq(SMatrix m1, SMatrix m2) {
    if (m1.cols != m2.cols || m1.rows != m2.rows) {
        return 0;
    }

    for (uint64_t i = 0; i < m1.rows; ++i) {
        for (uint64_t j = 0; j < m1.cols; ++j) {
            if (smatrix_at(m1, i, j) != smatrix_at(m2, i, j)) {
                return 0;
            }
        }
    }

    return 1;
}


int smatrix_is_skew_symmetric(SMatrix m) {
    SMatrix t = smatrix_transpose(m);
    smatrix_mult(m, -1.f);

    if (smatrix_eq(m, t)) {
        smatrix_free(&t);
        smatrix_mult(m, -1.f);
        return 1;
    }

    smatrix_free(&t);
    smatrix_mult(m, -1.f);
    return 0;
}


void smatrix_free(SMatrix *m) {
    m->cols = 0;
    m->rows = 0;
    m->elements = 0;

    free(m->rows_s);
    free(m->cols_i);
    free(m->values);
```

```
    m = NULL;
}


#endif // SPARSE_MATRIX_IMPLEMENTATIONvitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat makefile
CC = gcc
CFLAGS = -std=c99 -Wall -Wextra

main:
        $(CC) $(CFLAGS) -o main.out main.c
debug:
        $(CC) $(CFLAGS) -g -o main.out main.c
clean:
        rm -f *.o main.outvitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat m1.txt
0 2 0
-2 0 -0.5
0 0.5 0vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat m2.txt
3 -1.2 0
4 2  0
0 0  0
0 -1 0vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat m3.txt
0   -5.2 0 31 -2
5.2    0 0  2  0
0      0 0  0  0
-31   -2 0  0  0
2      0 0  0  0vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ make
gcc -std=c99 -Wall -Wextra -o main.out main.c
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ ./main.out
rows = 0, alloced rows_s = 2
rows = 1, alloced rows_s = 3
rows = 2, alloced rows_s = 4
Initial matrix:
(Real matrix)
[
   0.000    2.000    0.000
  -2.000    0.000   -0.500
   0.000    0.500    0.000
]
(Internal structure)
rows = 3, cols = 3, elements = 4
rows_s: [0 1 3 (4)]
cols_i: [1 2 0 1]
values: [2.000000 -0.500000 -2.000000 0.500000]
========================================
Transposed by secondary diagonal matrix:
(Real matrix)
[
   0.000   -0.500    0.000
   0.500    0.000    2.000
   0.000   -2.000    0.000
]
(Internal structure)
rows = 3, cols = 3, elements = 4
rows_s: [0 1 3 (4)]
cols_i: [1 0 2 1]
values: [-0.500000 0.500000 2.000000 -2.000000]

The matrix is skew-symmetric!
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat main.c
#define SPARSE_MATRIX_IMPLEMENTATION
#include "smatrix.h"


int main() {
    FILE *f = fopen("m2.txt", "r");
    assert(f != NULL);

    SMatrix m = smatrix_from_file(f);

    printf("Initial matrix:\n");
    printf("(Real matrix)\n");
    smatrix_print(m);
    printf("(Internal structure)\n");
    smatrix_print_debug(m);

    printf(
        "========================================\n"
        "Transposed by secondary diagonal matrix:\n"
    );
    SMatrix t = smatrix_transpose_sec_diag(m);
    printf("(Real matrix)\n");
    smatrix_print(t);
    printf("(Internal structure)\n");
    smatrix_print_debug(t);
```

```
    printf("\n");
    if (smatrix_is_skew_symmetric(t)) {
        printf("The matrix is skew-symmetric!\n");
    } else {
        printf("The matrix is not skew-symmetric...\n");
    }

    smatrix_free(&m);
    smatrix_free(&t);
    fclose(f);

    return 0;
}vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ make
gcc -std=c99 -Wall -Wextra -o main.out main.c
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ ./main.out
rows = 0, alloced rows_s = 2
rows = 1, alloced rows_s = 3
rows = 2, alloced rows_s = 4
rows = 3, alloced rows_s = 5
Initial matrix:
(Real matrix)
[
   3.000   -1.200    0.000
   4.000    2.000    0.000
   0.000    0.000    0.000
   0.000   -1.000    0.000
]
(Internal structure)
rows = 4, cols = 3, elements = 5
rows_s: [0 2 4 4 (5)]
cols_i: [1 0 1 0 1]
values: [-1.200000 3.000000 2.000000 4.000000 -1.000000]
========================================
Transposed by secondary diagonal matrix:
(Real matrix)
[
   0.000    0.000    0.000    0.000
  -1.000    0.000    2.000   -1.200
   0.000    0.000    4.000    3.000
]
(Internal structure)
rows = 3, cols = 4, elements = 5
rows_s: [0 0 3 (5)]
cols_i: [0 2 3 2 3]
values: [-1.000000 2.000000 -1.200000 4.000000 3.000000]

The matrix is not skew-symmetric...
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ cat main.c
#define SPARSE_MATRIX_IMPLEMENTATION
#include "smatrix.h"


int main() {
    FILE *f = fopen("m3.txt", "r");
    assert(f != NULL);

    SMatrix m = smatrix_from_file(f);

    printf("Initial matrix:\n");
    printf("(Real matrix)\n");
    smatrix_print(m);
    printf("(Internal structure)\n");
    smatrix_print_debug(m);

    printf(
        "========================================\n"
        "Transposed by secondary diagonal matrix:\n"
    );
    SMatrix t = smatrix_transpose_sec_diag(m);
    printf("(Real matrix)\n");
    smatrix_print(t);
    printf("(Internal structure)\n");
    smatrix_print_debug(t);

    printf("\n");
    if (smatrix_is_skew_symmetric(t)) {
        printf("The matrix is skew-symmetric!\n");
    } else {
        printf("The matrix is not skew-symmetric...\n");
    }

    smatrix_free(&m);
    smatrix_free(&t);
    fclose(f);
```

```
    return 0;
}vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ make
gcc -std=c99 -Wall -Wextra -o main.out main.c
vitos@vitos-hp16:/mnt/c/important/docs/mai/labs/lvii$ ./main.out
rows = 0, alloced rows_s = 2
rows = 1, alloced rows_s = 3
rows = 2, alloced rows_s = 4
rows = 3, alloced rows_s = 5
rows = 4, alloced rows_s = 6
Initial matrix:
(Real matrix)
[
   0.000   -5.200    0.000   31.000   -2.000
   5.200    0.000    0.000    2.000    0.000
   0.000    0.000    0.000    0.000    0.000
 -31.000   -2.000    0.000    0.000    0.000
   2.000    0.000    0.000    0.000    0.000
]
(Internal structure)
rows = 5, cols = 5, elements = 8
rows_s: [0 3 5 5 7 (8)]
cols_i: [4 3 1 3 0 1 0 0]
values: [-2.000000 31.000000 -5.200000 2.000000 5.200000 -2.000000 -31.000000 2.000000]
========================================
Transposed by secondary diagonal matrix:
(Real matrix)
[
   0.000    0.000    0.000    0.000   -2.000
   0.000    0.000    0.000    2.000   31.000
   0.000    0.000    0.000    0.000    0.000
   0.000   -2.000    0.000    0.000   -5.200
   2.000  -31.000    0.000    5.200    0.000
]
(Internal structure)
rows = 5, cols = 5, elements = 8
rows_s: [0 1 3 3 5 (8)]
cols_i: [4 3 4 1 4 0 1 3]
values: [-2.000000 2.000000 31.000000 -2.000000 -5.200000 2.000000 -31.000000 5.200000]

The matrix is skew-symmetric!
```

**9**. **Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|---|---|---|---|---|---|---|
| | | | | | | |

**10**. **Замечания автора** по существу работы: _____

_____

_____

_____

**11**. **Выводы:** Я смог реализовать одно из представлений разреженных матриц в памяти компьютера. Научился пользоваться модулем assert для простой обработки ошибок

_____

_____

Недочёты при выполнении задания могут быть устранены следующим образом: _____

_____

_____

_____

_____

Подпись студента _____