Московский авиационный институт

(Национальный исследовательский университет)

Курсовой проект

по курсу

«Языки и методы программирования»

2 семестр

Задание 7

Выполнил: Тимирчев Янис

Группа: М8О-111Б-21

Руководитель: Никулин С.П.

Оценка:

Дата:

Москва

2022г

Задание

Составить программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами вещественного типа, предусмотреть выполнение следующих стандартных функций.

Стандартные:

1. Ввод матрицы
2. Печать матрицы в обычном виде
3. Печать матрицы во внутреннем виде
4. Произвести операцию над матрицей
5. Выход из программы

Вариант физического хранения 2 - отображение на динамические структуры

Вариант схемы размещения 3 - три вектора

Вариант преобразования 3 - найти элемент матрицы, ближайший к заданному значению. Разделить на него элементы строки и столбца, на пересечении которых он расположен. Если таких элементов несколько, обработать все.

Структура и алгоритм проекта.

Проект состоит из 2 файлов: sparse_matrix_pointers.h и kp7.c.
Алгоритм следующий:
Сначала инициализируем три вектора, с помощью которых будет представлена матрица. Затем мы будем их заполнять. Реализуем функции для вывода матрицы в обычном виде и внутреннем виде. Реализуем функцию, которая находит ближайший к заданному элемент и делит на него нужные строку и столбец.
Напишем основную программу с меню, в котором можно выбирать нужное действие. Также сделаем проверку, чтобы введённый элемент не был нулём.

Программный код и тестовые данные.

sparse_matrix_pointers.h

```
#ifndef SPARSE_MATRIX_H
#define SPARSE_MATRIX_H

#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct SparseMx {
    int* (*CIP)[];

    int (*PI)[];

    int* pi_end;

    double (*YI)[];

    int N;
    int n_of_nonzero;
} SparseMx;

void mx_free(SparseMx* mx) {
    if (mx->n_of_nonzero == 0) return;

    free(mx->CIP);
    free(mx->PI);
    free(mx->YI);
    free(mx);
}

SparseMx* mx_new(int N, int n_of_nonzero) {
    SparseMx* mx = (SparseMx*)malloc(sizeof(SparseMx));

    mx->N = N;
    mx->n_of_nonzero = n_of_nonzero;

    if (!mx) {
        printf("Couldn't allocate SparseMatrix with n = %d\n", N);
```

```
        return NULL;
    }

    // init arrays
    mx->CIP = (int*(*)[])malloc(n_of_nonzero * sizeof(int*));
    mx->PI = (int(*)[])malloc(n_of_nonzero * sizeof(int));
    mx->YI = (double(*)[])malloc(n_of_nonzero * sizeof(double));

    mx->pi_end = &(*mx->PI)[n_of_nonzero - 1];

    if (!mx->CIP || !mx->PI || !mx->YI) {
        printf("Couldn't allocate SpaceMatrix's arrays\n");

        mx_free(mx);

        return NULL;
    }

    return mx;
}

void mx_read(FILE* fi, SparseMx* matrix) {
    int elem_i = 0;

    for (size_t row = 0; row < matrix->N; row++) {
        (*matrix->CIP)[row] = &(*matrix->PI)[elem_i];

        int elems_on_cur_row = 0;

        for (size_t column = 0; column < matrix->N; column++) {
            double val;
            fscanf(fi, " %lf", &val);

            if (val != 0) {
                (*matrix->PI)[elem_i] = column;
                (*matrix->YI)[elem_i] = val;

                elems_on_cur_row++;
                elem_i++;
            }
        }
```

```
    }
}

int get_pi_i(SparseMx* mx, int* pi_cur) { return pi_cur - &(*mx->PI)[0]; }

double mx_get(SparseMx* mx, int row, int column) {
    if (mx->n_of_nonzero == 0) return 0;

    if (row >= mx->N || column >= mx->N || row < 0 || column < 0) {
        printf("Index out of range. Returns 0\n");
        return 0;
    }

    int* cur_pi = (*mx->CIP)[row];
    int cur_column = *cur_pi;
    int* next_pi = NULL;

    if (row <= (mx->N - 2)) next_pi = (*mx->CIP)[row + 1];

    while (true) {
        if (cur_pi == next_pi) return 0;
        if (*cur_pi == column) {
            int i = get_pi_i(mx, cur_pi);
            return (*mx->YI)[i];
        };

        if (cur_pi == mx->pi_end) return 0;

        cur_pi++;
    }
}

void mx_print_friendly(SparseMx* matrix) {
    printf("Full form of matrix:\n");
    for (size_t row = 0; row < matrix->N; row++) {
        for (size_t column = 0; column < matrix->N; column++) {
            printf("%.2f ", mx_get(matrix, row, column));
        }
        printf("\n");
    }
    printf("\n");
}
```

```
void mx_print_inner(SparseMx* matrix) {
    printf("Internal representation of matrix:\n");
    printf("1. CIP - points to PI:\t");
    for (size_t i = 0; i < matrix->N; i++) {
        printf("%d ", *(*matrix->CIP)[i]);
    }
    printf("\n");

    printf("2. PI - columnumns:\t");
    for (size_t i = 0; i < matrix->n_of_nonzero; i++) {
        printf("%d ", (*matrix->PI)[i]);
    }
    printf("\n");

    printf("3. YI - values:\t\t");
    for (size_t i = 0; i < matrix->n_of_nonzero; i++) {
        printf("%.2f ", (*matrix->YI)[i]);
    }
    printf("\n");
    printf("\n");
}

void mx_divide_column_by(SparseMx* mx, int column, double val) {
    for (size_t i = 0; i < mx->n_of_nonzero; i++) {
        int cur_column = (*mx->PI)[i];

        if (cur_column == column) {
            (*mx->YI)[i] = (*mx->YI)[i] / val;
        }
    }
}

void mx_divide_row_by(SparseMx* mx, int row, double val, int exclude_column) {
    // pointer to start of row in PI
    int* pi_cur = (*mx->CIP)[row];

    // pointer to start of next row in PI
    int* pi_next = NULL;
    if (row < mx->N - 1) {
        pi_next = (*mx->CIP)[row + 1];
    }
```

```
    while (pi_cur != pi_next) {
        // do not divide twice
        int i = get_pi_i(mx, pi_cur);
        if (*pi_cur != exclude_column) (*mx->YI)[i] = (*mx->YI)[i] / val;

        if (pi_cur == mx->pi_end) break;

        pi_cur++;
    }
}

void mx_divide_by_nearest_val(SparseMx* mx, double x) {
    if (mx->n_of_nonzero == 0) {
        printf("Matrix has 0 nonzero elements\n");
        return;
    }

    // find nearest value
    double delta = abs(x - (*mx->YI)[0]);
    double nearest = (*mx->YI)[0];

    for (size_t i = 1; i < mx->n_of_nonzero; i++) {
        double val = (*mx->YI)[i];
        double cur_delta = abs(x - val);

        if (cur_delta < delta) {
            delta = cur_delta;
            nearest = val;
        }
    }

    if (nearest == 0) {
        printf("Nearest value is 0. Cannot divide by zero\n");
        return;
    }

    int* cross_rows = (int*)malloc(mx->n_of_nonzero * sizeof(int));
    int* cross_columnumns = (int*)malloc(mx->n_of_nonzero * sizeof(int));
    cross_rows[0] = -1;
    cross_columnumns[0] = -1;
    int found_crosses = 0;
```

```
for (size_t row = 0; row < mx->N; row++) {
    int* pi_cur = (*mx->CIP)[row];

    int* pi_next = NULL;
    if (row < mx->N - 1) {
        pi_next = (*mx->CIP)[row + 1];
    }

    // iterate all nonzero elements of the row

    while (pi_cur != pi_next) {
        int i = get_pi_i(mx, pi_cur);
        double val = (*mx->YI)[i];
        int column = *pi_cur;

        // push found cross (row and column) to the array
        if (val == nearest) {
            cross_rows[found_crosses] = row;
            cross_columnumns[found_crosses] = column;

            found_crosses++;
            if (found_crosses != mx->n_of_nonzero) {
                cross_rows[found_crosses] = -1;
                cross_columnumns[found_crosses] = -1;
            }
        }

        if (pi_cur == mx->pi_end) break;

        pi_cur++;
    }
}

for (size_t i = 0; i < mx->n_of_nonzero; i++) {
    if (cross_columnumns[i] == -1) break;

    mx_divide_column_by(mx, cross_columnumns[i], nearest);
}

for (size_t i = 0; i < mx->n_of_nonzero; i++) {
    if (cross_rows[i] == -1) break;
```

```
        mx_divide_row_by(mx, cross_rows[i], nearest, cross_columnumns[i]);
    }
}

#endif
```

kp7.c

```
#include <stdio.h>

#include "sparse_matrix_pointers.h"

int main(int argc, char *argv[]) {
    int input = -1;

    SparseMx *matrix = NULL;

    printf(
        "Enter a command:\n    1 - read matrix;\n    2 - print matrix in a "
        "userfriendly way;\n    3 - "
        "print matrix in internal representation;\n    4 - action: find "
        "nearest value and divide all respective rows and columnumns by this value;"
        "range\n");
    while (scanf("%d", &input) != EOF) {
        switch (input) {
            // read matrix
            case 1: {
                printf("Enter N, n_of_nonzero: ");

                int n, n_of_nonzero;
                scanf(" %d", &n);
                scanf(" %d", &n_of_nonzero);

                if (n < 0 || n_of_nonzero < 0) {
                    printf(
                        "N should be at least 0, n_of_nonzero should be at "
                        "least 0");
                    break;
                }

                matrix = mx_new(n, n_of_nonzero);
```

```
    printf("Enter the matrix: \n");

    mx_read(stdin, matrix);

    break;
}

// print matrix in a userfriendly way
case 2:
    if (matrix == NULL) {
        printf("Matrix was not readen\n");
        break;
    }
    mx_print_friendly(matrix);
    break;

// print matrix in internal representation
case 3: {
    if (matrix == NULL) {
        printf("Matrix was not read\n");
        break;
    }

    mx_print_inner(matrix);

    break;
}

// action
case 4: {
    if (matrix == NULL) {
        printf("Matrix was not read\n");
        break;
    }

    int p;
    printf("Enter p: ");
    scanf(" %d", &p);

    if (p == 0) {
        printf("p should not be zero\n");
```

```
            }

            mx_divide_by_nearest_val(matrix, p);

            printf("\n");

            break;
        }

        default:
            printf("Incorrect input.\n");

            break;
    }

    printf("Enter a command:\n");
  }

  mx_free(matrix);
}
```

Тестирование:

```
ianis@dev11:~/Documents$ gcc kp7.c -o kp7
ianis@dev11:~/Documents$./kp7
Enter a command:
   1 - read matrix;
   2 - print matrix in a userfriendly way;
   3 - print matrix in internal representation;
   4 - action: find nearest value and divide all respective rows and columnumns by this value;
1
Enter N, n_of_nonzero: 6 8
Enter the matrix:
1 3 7 0 0 0
0 0 0 3 0 0
5 0 0 0 0 0
0 0 0 0 0 0
0 8 0 9 0 0
0 3 0 0 0 0
Enter a command: 2
Full form of matrix:
1.00 3.00 7.00 0.00 0.00 0.00
```

```
0.00 0.00 0.00 3.00 0.00 0.00
5.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 8.00 0.00 9.00 0.00 0.00
0.00 3.00 0.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI:  0 3 0 1 1 1
2. PI - columnumns:        0 1 2 3 0 1 3 1
3. YI - values:          1.00 3.00 7.00 3.00 5.00 8.00 9.00 3.00

Enter a command: 4
Enter p: 3
Enter a command: 2
Full form of matrix:
0.33 0.33 2.33 0.00 0.00 0.00
0.00 0.00 0.00 1.00 0.00 0.00
5.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.89 0.00 3.00 0.00 0.00
0.00 0.33 0.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI:  0 3 0 1 1 1
2. PI - columnumns:        0 1 2 3 0 1 3 1
3. YI - values:          0.33 0.33 2.33 1.00 5.00 0.89 3.00 0.33

Enter a command: 1
Enter N, n_of_nonzero: 4 0
Enter the matrix:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

Enter a command: 2
Full form of matrix:
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
```

```
0.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI:  0 0 0 0
2. PI - columns:
3. YI - values:

Enter a command: 4
Enter p: 3
Matrix has 0 nonzero elements
Enter a command: 2
Full form of matrix:
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI:  0 0 0 0
2. PI - columnumns:
3. YI - values:

Enter a command: 4
Enter p: 0
p should not be zero
Matrix has 0 nonzero elements
Enter a command: 1
Enter N, n_of_nonzero: 6 7
Enter the matrix:
0 0 20 2 0 3
0 0 0 0 0 0
2 0 2 0 0 0
0 0 0 10.5 0 0
0 0 0 0 0 0
0 0 10 0 0 0

Enter a command: 2
Full form of matrix:
0.00 0.00 20.00 2.00 0.00 3.00
0.00 0.00 0.00 0.00 0.00 0.00
```

2.00 0.00 2.00 0.00 0.00 0.00
0.00 0.00 0.00 10.50 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 10.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI: 2 0 0 3 2 2
2. PI - columnumns: 2 3 5 0 2 3 2
3. YI - values: 20.00 2.00 3.00 2.00 2.00 10.50 10.00

Enter a command: 4
Enter p: 2
Enter a command: 2
Full form of matrix:
0.00 0.00 5.00 1.00 0.00 1.50
0.00 0.00 0.00 0.00 0.00 0.00
0.50 0.00 0.50 0.00 0.00 0.00
0.00 0.00 0.00 5.25 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 5.00 0.00 0.00 0.00

Enter a command: 3
Internal representation of matrix:
1. CIP - points to PI: 2 0 0 3 2 2
2. PI - columnumns: 2 3 5 0 2 3 2
3. YI - values: 5.00 1.00 1.50 0.50 0.50 5.25 5.00

Заключение.

Я составил программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами вещественного типа, предусмотрев выполнение следующих стандартных функций.