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

Студент группы М8О-103Б-22 Ахметшин Булат Рамилевич, № по списку 2

Контакты www, e-mail, icq, skype ahmbulat04@yandex.ru

Работа выполнена: 05.05.2023 г.

Преподаватель: доцент каф. 806 Никулин С.П.

Входной контроль знаний с оценкой _____

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

Подпись преподавателя _____

1. **Тема:** Разреженные матрицы. _____

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

3. **Задание** (*вариант № 2 - {2, 2, 2}*): Определить максимальный по модулю элемент матрицы и разделить на него все элементы столбца, в котором он находится. Если таких элементов несколько, обработать предпоследний стобец, содержащий такой элемент. _____

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(R) Core(TM) i7-10510U____ с ОП 8 ГБ____ НМД SSD 512 ГБ____ . Монитор Встроенный 1920x1080 _____
Другие устройства _____

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 Ubuntu_____ версия 22.04_____
интерпретатор команд GNU bash_____ версия 5.1.16_____
Система программирования Visual Studio Code_____ версия 1.77.3_____
Редактор текстов Sublime Text 3_____ версия 3211_____
Утилиты операционной системы Стандартные утилиты OS Linux_____

Прикладные системы и программы Редактор текста nano.
Местонахождение и имена файлов программ и данных на домашнем компьютере
/home/bulat/Studying/prprm/cr/VII/

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

Для хранения разреженной матрицы будет использована структура из одностороннего списка, искомого максимального по модулю элемента и соответствующего ему столбца.
Поля этой струкуры будут определяться во время считывания матрицы.
Делиться элементы соответствующего столбца будут в последовательности, в которой они внесены в список.

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

Составить makefile, написать реализацию линейного одностороннего списка и и функции считывания матрицы, выполнения соответствующих вычислений и печати разреженной матрицы во внешнем и внутреннем представлениях.

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)

```
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ ls
list.c  list.h  logs  main.c  makefile  test1  test2  test3
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ cat makefile
CC = gcc
CFLAGS = -std=c99 -Wall -Wextra

main: main.o list.o
    $(CC) $(CFLAGS) -o main main.o list.o
main_o:
    $(CC) $(CFLAGS) -c main.c
deque_o:
    $(CC) $(CFLAGS) -c list.c
clean:
    rm -f *.o main
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ cat list.h
#ifndef _LIST_H
#define _LIST_H

#include <stdbool.h>
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

typedef uint64_t size_t;

typedef struct {
    char* p;
    size_t l;
} string;

string make_string(char* const);
bool strings_eq(string, string);

typedef struct node {
    struct node* r;
    double x;
} node;

node* make_node(double);

void link_node(node*, node*, node*);

void link_nodes(node*, node*);

void node_free(node*);

typedef struct {
    node* begin;
    node* back;
    size_t size;
} list;

void list_create_empty(list*);

list list_copy(list* const);

void list_add(list*, double, size_t);

node* list_find_node(list* const, double);

void list_remove(list*, double);

void list_remove_subtree(list*, node*, node*);

void list_free(list*);

size_t list_len(node* const, node* const);

void list_print(list* const);

void list_cut(list*, char, double);


#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define abs(x) ((x) > 0 ? (x) : (-x))
```

```
#endifbulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ cat list.c
#include "list.h"

string make_string(char* const p) {
    string s;
    s.p = p;
    s.l = 0;
    while (s.p[s.l] != '\0') {
        ++s.l;
    }
    return s;
}

bool string_eq(string a, string b) {
    if (a.l != b.l) {
        return false;
    }
    size_t i = 0;
    while (i < a.l) {
        if (a.p[i] != b.p[i]) {
            return false;
        }
        ++i;
    }
    return true;
}

node* make_node(double x) {
    node* p = (node*)malloc(sizeof(node));
    p->r = NULL;
    p->x = x;
    return p;
}

void link_node(node* l, node* m, node* r) {
    if (! (l == NULL))
        l->r = m;
    m->r = r;
}

void link_nodes(node* l, node* r) {
    if (! (l == NULL)) {
        l->r = r;
    }
}

void node_free(node* p) {
    p->r = NULL;
    p->x = 0;
}

void list_create_empty(list* l) {
    l->begin = l->back = NULL;
    l->size = 0;
}

list list_copy(list* const l) {
    list _l;
    if (l->size == 0) {
        return _l;
    }
    _l.begin = make_node(l->begin->x);
    node* _p = _l.begin;
    node* p = l->begin->r;
    while (true) {
        _p->r = make_node(p->x);
        p = p->r;
        _p = _p->r;
        if (p == NULL) {
            break;
        }
    }
    _l.back = _p;
    _l.size = l->size;
    return _l;
}

void list_add(list* l, double x, size_t index) {
    node* _p = make_node(x);
    if (index < l->size) {
        node* p = l->begin;
        for (size_t i = 0; i < index; ++i) {
            p = p->r;
        }
        link_node(p, _p, p->r);
```

```
    } else {
        if (l->back == NULL) {
            l->begin = l->back = _p;
            l->size = 0;
        } else {
            link_node(l->back, _p, NULL);
            l->back = _p;
        }
    }
    l->size++;
}

node* list_find_node(list* const l, double x) {
    node* p = l->begin;
    while (! (p == NULL) && p->x != x) {
        p = p->r;
    }
    return p;
}

void list_remove(list* l, double x) {
    if (! (l->begin == NULL) && l->begin->x == x) {
        node* r = l->begin;
        l->begin = l->begin->r;
        if (l->begin == NULL) {
            l->back = NULL;
        } else {
            link_nodes(l->begin, l->begin->r->r);
        }
        node_free(r);
        l->size--;
    }
    else {
        node* p = l->begin;
        while (!(p->r == NULL) && p->r->x != x) {
            p = p->r;
        }
        if (! (p->r == NULL)) {
            node* r = p->r;
            link_nodes(p, p->r->r);
            node_free(r);
            l->size--;
        }
    }
}

void list_remove_subtree(list* l, node* s, node* e) {
    node* p = l->begin,* r;
    if (p != s) {
        while (! (p->r == NULL) && ! (p->r == s)) {
            p = p->r;
        }
        if (p->r == NULL) {
            return;
        }
        if (e == NULL) {
            l->back = p;
        }
        r = p->r;
        p->r = e;
        p = r;
    } else {
        l->begin = e;
    }
    while (! (p->r == NULL) && ! (p->r == e)) {
        r = p;
        p = p->r;
        node_free(r);
        --l->size;
    }
    node_free(p);
    --l->size;
}

void list_free(list* l) {
    if (l->begin == NULL) {
        l->back = NULL;
        l->size = 0;
        return;
    }
    node* p = l->begin,* r;
    while (! (p->r == NULL)) {
        r = p;
        p = p->r;
        node_free(r);
```

```
    }
    node_free(p);
    l->begin = l->back = NULL;
    l->size = 0;
}

size_t list_len(node* const l, node* const r) {
    if (l == NULL || r == NULL) {
        return 0;
    }
    node* p = l;
    size_t len = 1;
    while (! (p == r)) {
        p = p->r; ++len;
    }
    return len;
}

void list_print(list* const l) {
    printf("{");
    if (! (l->begin == NULL)) {
        node* p = l->begin;
        while (! (p->r == NULL)) {
            if (abs(p->x - (int64_t)(p->x)) < 1e-3) {
                printf(" %ld,", (int64_t)(p->x));
            } else {
                printf(" %.3f,", p->x);
            }
            p = p->r;
        }
        if (p->x - (int64_t)(p->x) < 1e-3) {
            printf(" %ld ", (int64_t)(p->x));
        } else {
            printf(" %.3f ", p->x);
        }
    }
    printf("}");
}

void list_cut(list* l, char side, double x) {
    node* p = list_find_node(l, x);
    if (p == NULL) {
        return;
    }
    if (side == 'l' && ! (p == l->begin)) {
        list_remove_subtree(l, l->begin, p);
    } else if (side == 'r' && ! (p->r == NULL)) {
        list_remove_subtree(l, p->r, NULL);
    }
}bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ cat main.c
#include <inttypes.h>
#include <stdbool.h>
#include <stdio.h>
#include <ctype.h>

#include "list.h"


typedef struct {
    list l;
    double mxe;
    size_t ind;
} sparce_matrix;

void read_sparse_matrix_from_file(sparce_matrix* mx, const char* fname) {
    FILE* f = fopen(fname, "r");
    if (f == NULL) {
        printf("Unable to open file %s", fname);
        exit(-1);
        return;
    }

    mx->mxe = 0;
    list* l = &(mx->l);
    list_create_empty(l);
    list_add(l, 0, l->size);
    list_add(l, 1, l->size);
    char c = '~';
    int64_t i = 1, j = 0;

    double x = 0;

    size_t fraction = 0;
    int8_t multiplier = 1;
```

```
    int64_t last_column = -1, penult_column = -1;

    while (true) {
        c = fgetc(f);

        if (c == ' ') {
            multiplier = 1;
            fraction = 0;
            list_add(l, j + 1, l->size);
            list_add(l, x, l->size);
            x = 0;
            ++j;
        } else if (c == '\n') {
            multiplier = 1;
            fraction = 0;
            list_add(l, j + 1, l->size);
            list_add(l, x, l->size);
            list_add(l, 0, l->size);
            list_add(l, i + 1, l->size);
            x = 0;
            ++i; j = 0;
        } else if (c == EOF) {
            list_add(l, j + 1, l->size);
            list_add(l, x, l->size);
            break;
        } else if (isdigit(c)) {
            if (fraction) {
                x += (double)(c - 48) / fraction * multiplier;
                fraction *= 10;
            } else {
                x = 10 * x + (c - 48) * multiplier;
            }
        } else if (c == '.') {
            fraction = 10;
        }  else if (c == '-') {
            multiplier = -1;
        }
        if (abs(mx->mxe) < abs(x)) {
            last_column = j;
            penult_column = -1;
            mx->mxe = x;
            mx->ind = j;
        } else if (abs(abs(mx->mxe) - abs(x)) < 1e-7) {
            if (last_column < j) {
                if (-1 < penult_column) {
                    mx->mxe = x;
                    mx->ind = last_column;
                }
                penult_column = last_column;
                last_column = j;
            } else if (penult_column < j && j < last_column) {
                penult_column = j;
                mx->mxe = x;
                mx->ind = penult_column;
            }
        }
    }

    list_add(l, 0, l->size);
    list_add(l, 0, l->size);

    fclose(f);
}

void print_sparse_matrix(list* l) {
    node* p = l->begin;
    if (p == NULL) {
        return;
    }
    int64_t j = 0;
    while (true) {
        if (p->x == 0) {
            if (! (p == l->begin)) {
                printf("\n");
            }
            j = 0;
            if (p->r->x == 0) {
                break;
            }
            p = p->r;
        } else {
            while (j + 1 < (int64_t)(p->x)) {
                printf("0\t");
                ++j;
            }
```

```
            p = p->r; ++j;
            if (abs(p->x - (int64_t)(p->x)) < 1e-3) {
                printf(" %ld\t", (int64_t)(p->x));
            } else {
                printf(" %.3f\t", p->x);
            }
        }
        p = p->r;
    }
}

sparce_matrix calc_func(sparce_matrix mx) {
    sparce_matrix _mx;
    _mx.l = list_copy(&(mx.l));
    _mx.mxe = mx.mxe;
    _mx.ind = mx.ind;

    node* p = _mx.l.begin;
    while (true) {
        if (p->x == 0) {
            if (p->r->x == 0) {
                break;
            }
            p = p->r;
        } else {
            if (p->x == _mx.ind + 1) {
                p = p->r;
                p->x /= _mx.mxe;
            } else {
                p = p->r;
            }
        }
        p = p->r;
    }
    return _mx;
}

int main(int arc, char** argv) {

    sparce_matrix mx;

    for (int i = 1; i < arc; ++i) {
        printf("File: %s\n\n", argv[i]);

        read_sparse_matrix_from_file(&mx, argv[i]);

        printf("Origin matrix:\n");
        print_sparse_matrix(&(mx.l));
        printf("Internal representation:\n");
        list_print(&(mx.l));

        printf("\n" );

        sparce_matrix _mx = calc_func(mx);

        printf("Calculated matrix:\n");
        print_sparse_matrix(&(_mx.l));
        printf("Internal representation:\n");
        list_print(&(_mx.l));
        printf("\n");

        list_free(&(mx.l));
        list_free(&(_mx.l));
        printf("\n\n");
    }




    return 0;
}
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ make
gcc -std=c99 -Wall -Wextra   -c -o main.o main.c
gcc -std=c99 -Wall -Wextra   -c -o list.o list.c
gcc -std=c99 -Wall -Wextra -o main main.o list.o
bulat@bulat-Swift-SF314-58:~/Studying/prprm/cr/VII$ ./main test1 test2 test3
File: test1

Origin matrix:
 0   -5.200  0   5   -2.000
 5.200   0   0   2   0
 0   0   0   0   0
 -31.000     -2.000  0   0   0
 2   0   31  0   0
Internal representation:
```

```
{ 0, 1, 1, 0, 2, -5.200, 3, 0, 4, 5, 5, -2.000,
  0, 2, 1, 5.200, 2, 0, 3, 0, 4, 2, 5, 0,
  0, 3, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0,
  0, 4, 1, -31.000, 2, -2.000, 3, 0, 4, 0, 5, 0,
  0, 5, 1, 2, 2, 0, 3, 31, 4, 0, 5, 0, 0, 0 }
Calculated matrix:
 0   -5.200  0   5   -2.000
 -0.168  0   0   2   0
 0   0   0   0   0
 1   -2.000  0   0   0
 -0.065  0   31  0   0
Internal representation:
{ 0, 1, 1, 0, 2, -5.200, 3, 0, 4, 5, 5, -2.000,
  0, 2, 1, -0.168, 2, 0, 3, 0, 4, 2, 5, 0,
  0, 3, 1, 0, 2, 0, 3, 0, 4, 0, 5, 0,
  0, 4, 1, 1, 2, -2.000, 3, 0, 4, 0, 5, 0,
  0, 5, 1, -0.065, 2, 0, 3, 31, 4, 0, 5, 0, 0, 0 }


File: test2

Origin matrix:
 3   -1.200  0
 4   2   0
 0   0   0
 0   -1.000  0
Internal representation:
{ 0, 1, 1, 3, 2, -1.200, 3, 0,
  0, 2, 1, 4, 2, 2, 3, 0,
  0, 3, 1, 0, 2, 0, 3, 0,
  0, 4, 1, 0, 2, -1.000, 3, 0, 0, 0 }
Calculated matrix:
 0.750   -1.200  0
 1   2   0
 0   0   0
 0   -1.000  0
Internal representation:
{ 0, 1, 1, 0.750, 2, -1.200, 3, 0,
  0, 2, 1, 1, 2, 2, 3, 0,
  0, 3, 1, 0, 2, 0, 3, 0,
  0, 4, 1, 0, 2, -1.000, 3, 0, 0, 0 }


File: test3

Origin matrix:
 0   2   0
 -2.000  0   -0.500
 0   0.500   0
Internal representation:
{ 0, 1, 1, 0, 2, 2, 3, 0,
  0, 2, 1, -2.000, 2, 0, 3, -0.500,
  0, 3, 1, 0, 2, 0.500, 3, 0, 0, 0 }
Calculated matrix:
 0   2   0
 1   0   -0.500
 0   0.500   0
Internal representation:
{ 0, 1, 1, 0, 2, 2, 3, 0,
  0, 2, 1, 1, 2, 0, 3, -0.500,
  0, 3, 1, 0, 2, 0.500, 3, 0, 0, 0 }
```

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:** в ходе этой лабораторной работы я получил опыт реализации линейного одностороннего списка и работы с разреженными матрицами.

12. Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента _____