Отчет по курсовому проекту N 7 по курсу

"Фундаментальная информатика"

Студент группы: М80-107Б-21, Павлов Иван Дмитриевич
Контакты: pavlov.id.2003@gmail.com
Работа выполнена: 22.04.2022
Преподаватель: Найденов Иван Евгеньевич

1 Тема

Разреженные матрицы

2 Цель работы

Составить программу на языке Си для обработки прямоугольных матриц

3 Задание

Реализовать разреженную матрицу (один вектор). Определить максимальный по модулю элемент матрицы и разделить на него все элементы строки, в которой он находится. Если таких элементов несколько, обработать каждую строку, содержащую такой элемент.

4 Оборудование

Процессор: AMD Ryzen 5 4600H with Radeon Graphics
ОП: 7851 Мб
НМД: 256 Гб
Монитор: 1920x1080

5 Программное обеспечение

Операционная система семейства: linux (ubuntu), версия 20.04.3 LTS
Интерпретатор команд: bash, версия 5.0.17(1)-release.
Система программирования: gcc*, версия 17
Редактор текстов: emacs, версия 25.2.2
Утилиты операционной системы: subl, make
Прикладные системы и программы: sublime text, bash
Местонахождение и имена файлов программ и данных на домашнем компьютере: /home/ggame/newlabs/

6 Идея, метод, алгоритм решения задачи

6.1 Разреженная матрица

Для реализации разреженной матрицы необходимы структуры: matrix, содержащая вектор и размеры матрицы, element, содержащая значение элемента матрицы, его строку и столбец и vector - динамический массив элементов. Для структуры вектор определены функции *void* v_create(), создающая пустой вектор; *void* v_push_back(), добавляющая новое значение в конец; *int* v_size(), которая возвращает размер вектора.
matrix m_create(int n, int m) - возврващает матрицу, согласно заданным размерам. Алгоритм заполнения: За $O(n^2)$ считываем элементы матрицы, если элемент не равен 0, то записываем в вектор из структуры матрицы значение, строку и столбец.
void m_print(matrix *A) - выводит разреженную матрицу на экран: запускаем двойной цикл и задаем счетчик *it*; если значение строки и столбца из вектора совпадают с счетчиками цикла, то выводим элемент вектора, соответствующий индексу *it*. Иначе выводим 0.

6.2 Функция варианта №1

Для реализации задания необходимы 3 функции:

- Поиск максимального элемента: обычный поиск максимума, только создаем элемент и проходим по вектору с $i \mathrel{+}= 3$ и присваиваем структуре значение, строку и столбец матрицы из вектора;
- Деление строки на элемент: также проходим вектор с $i \mathrel{+}= 3$ и проверяем, что элемент вектора, содержащий номер строки совпадает с нужной строкой; если да то делим;
- Совмещение этих двух функций: ищем в векторе все максимальные элементы и делим. Работает за $O(n^2)$

7 Сценарий выполнения работы

Листинг 1: test1

```
gggame@ggame:~/newlabs/kp7_r$ ./main
Enter count of rows & columns: 10 10
0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 34 0 0 0 0 17 0 0 0
0 34 0 0 0 0 17 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 34
0 0 0 0 0 0 0 0 0 33

0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 33
```

8 Распечатка протокола

Листинг 2: matrix.c

```c
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>

void v_create(vector *d)
{
  d->allocated = 1;
  d->size = 0;
  d->begin = malloc(d->allocated * sizeof(int));
}

void v_push_back(vector *d, int value)
{
  if (d->size + 1 >= d->allocated) {
    d->allocated *= 2;
    d->begin = realloc(d->begin, sizeof(int) * d->allocated + 1);
  }
  d->begin[d->size++] = value;
}
```

```
20
21      int v_size(vector *d)
22      {
23        return d->size;
24      }
25
26      matrix m_create(int n, int m)
27      {
28        matrix A;
29        int val;
30        v_create(&(A.matrix));
31
32        A.n = n;
33        A.m = m;
34
35        for (int i = 0; i < n; i ++) {
36          for (int j = 0; j < m; j ++) {
37            scanf("%d", &val);
38            if (val != 0) {
39              v_push_back(&(A.matrix), val);
40              v_push_back(&(A.matrix), i);
41              v_push_back(&(A.matrix), j);
42            }
43          }
44        }
45
46        return A;
47      }
48
49      void v_print(matrix *A) {
50        for (int i = 0; i < A->matrix.size; ++i) {
51          printf("%d ", A->matrix.begin[i]);
52        }
53        printf("\n");
54      }
55
56      void m_print(matrix *A)
57      {
58        int it = 0;
59        for (int i = 0; i < A->n; i ++) {
60          for (int j = 0; j < A->m; j ++) {
61            if (A->matrix.begin[it + 1] == i && A->matrix.begin[it + 2] == j) {
62              printf("%d ", A->matrix.begin[it]);
63              it += 3;
64            } else {
65              printf("0 ");
66            }
67          }
68          printf("\n");
69        }
70      }
71
72      int abs(int a)
73      {
74        if (a < 0) {
75          return -a;
76        }
77        return a;
78      }
79
80      element m_find_elem(matrix *A)
81      {
82        element el = {0, 0, 0};
83        int max = 0;
84
85        for (int i = 0; i < v_size(&(A->matrix)); i += 3) {
```

```
86          if (abs(A->matrix.begin[i]) > max) {
87            el.val = A->matrix.begin[i];
88            el.i = A->matrix.begin[i + 1];
89            el.j = A->matrix.begin[i + 2];
90            max = abs(A->matrix.begin[i]);
91          }
92        }
93
94        return el;
95      }
96
97      void m_div_str(matrix *A, int string, int div)
98      {
99        for (int i = 0; i < v_size(&(A->matrix)); i += 3) {
100         if (A->matrix.begin[i + 1] == string) {
101           A->matrix.begin[i] /= div;
102         }
103       }
104     }
105
106     int m_div_elem(matrix *A, int val)
107     {
108       element el_div = m_find_elem(A);
109
110       for (int i = 0; i < v_size(&(A->matrix)); i += 3) {
111         if (A->matrix.begin[i] == el_div.val && A->matrix.begin[i + 1] != el_div.i) {
112           m_div_str(A, A->matrix.begin[i + 1], el_div.val);
113         }
114       }
115       m_div_str(A, el_div.i, el_div.val);
116     }
```

Листинг 3: matrix.h

```
1       #ifndef __MATRIX_H__
2       #define __MATRIX_H__
3
4       typedef struct {
5         int *begin;
6         int size;
7         int allocated;
8       } vector;
9
10      typedef struct {
11        int val;
12        int i;
13        int j;
14      } element;
15
16      typedef struct
17      {
18        vector matrix;
19        int n, m;
20      } matrix;
21
22
23      void v_create(vector *d);
24
25      void v_push_back(vector *d, int value);
26
27      int v_size(vector *d);
28
29      matrix m_create(int n, int m);
30
31      int m_diag(matrix *A);
32
```

```
33      int m_div_elem(matrix *A, int val);
34
35      void m_print(matrix *A);
36
37      void v_print(matrix *A);
38
39      #endif
```

Листинг 4: main.c

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include "matrix.h"
4
5     int main(int argc, char const *argv[])
6     {
7       int n, m, val;
8       printf("Enter count of rows & columns: ");
9       scanf("%d%d", &n, &m);
10      matrix A = m_create(n, m);
11      printf("\n");
12      m_div_elem(&A, val);
13      m_print(&A);
14      return 0;
15    }
```

9 Вывод

Благодаря данному КП я научился работать с разреженной матрицей и экономить память.