Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего профессионального образования

«Московский авиационный институт»

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

2 семестр

Курсовой проект

По курсу «Алгоритмы и структуры данных»

Задание VII

Работ сдал: студент группы М8О-204Б-22

Филиппов Фёдор Иванович

Работу принял: преподаватель информатики

Потенко Максим Алексеевич

Москва, 2023

Оглавление

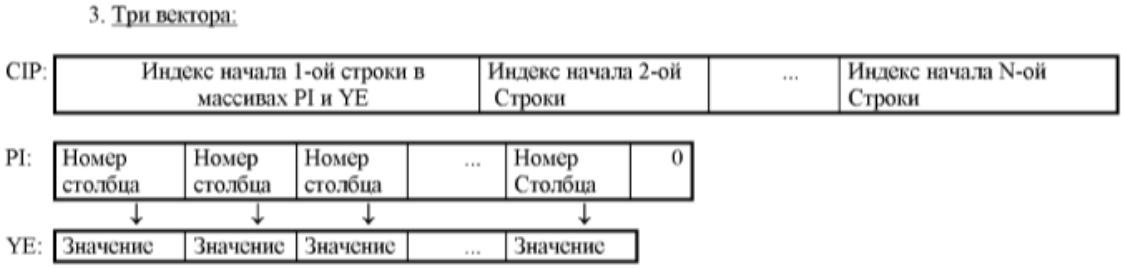
Цель работы

Составить программу на языке Си с процедурами и/или функциями для обработки прямоугольных матриц с элементами вещественного типа, которая:

1. Вводит матрицы для различного размера, представленные во входном текстовом файле в обычном формате, с одновременном размещением ненулевых элементов в разряженной матрице в соответствии со схемой
2. Печатает введенные матрицы во внутреннем представлении согласно заданной схеме размещения и в обычном виде
3. Выполняет необходимые преобразования разряженных матриц путем обращения к соответствующим процедурам и функциям
4. Печатает результат преобразования согласно заданной схеме размещения и в обычном виде

Задание

Схема размещения матрицы: Три вектора.



Преобразование: Для всех студентов, имеющих более одного компьютера, распечатать сведения о самом мощном из них.

Теоретическая справка

В данной курсовой работе мне предстоит изучить новый способ хранения матриц, в частности разреженных. Это будет осуществлено с использованием такой структуры, как вектор.

Вектор в программмировании

В программировании вектор — это структура данных, представляющая собой динамический массив элементов, которые хранятся в последовательном порядке в памяти. Вектор позволяет хранить и обрабатывать коллекцию элементов одного типа, при этом обеспечивая эффективный доступ к элементам по индексу и возможность изменения размера.

Именно это основная особенность вектора - его динамичность. Это означает, что размер вектора может быть изменен во время выполнения программы, в отличие от статического массива, размер которого определяется на этапе компиляции. При добавлении новых элементов вектор может автоматически расширяться, а при удалении - сжиматься, чтобы эффективно использовать память.

Доступ к элементам вектора осуществляется по индексу, начиная с 0. Это позволяет получать или изменять значения элементов по их позиции в векторе. Кроме того, векторы часто предоставляют различные методы и функции для выполнения операций, таких как сортировка, поиск, фильтрация и другие манипуляции с данными.

Векторы предоставляют гибкость и удобство в работе с коллекциями данных, позволяя эффективно хранить и обрабатывать большие объемы информации.

В языке С вектор может быть реализован с использованием динамической памяти и указателей.

Хранение разреженной матрицы

Хранение разреженной матрицы в разреженном строчном формате (CSR или CRS)

Разреженные матрицы являются основной составляющей многих приложений в области науки о данных, машинного обучения и компьютерной графики. Они применяются в таких областях, где матрицы содержат большое число нулей и занимают много памяти. Для эффективной работы с разреженными матрицами необходимо выбрать подходящий формат хранения, который позволит сократить расход памяти и обеспечить быстрый доступ к элементам матрицы. Одним из наиболее распространенных форматов хранения является разреженный строчный формат (CSR или CRS).

Разреженный строчный формат (CSR) представляет собой компактное представление разреженной матрицы, оптимизированное для операций сложения, умножения и обращения. Хранение происходит в виде трех массивов:

1. Массив значений (values): содержит ненулевые элементы матрицы в порядке их прохождения.
2. Массив колонок (columns): содержит индексы столбцов для каждого ненулевого элемента.
3. Массив индексов строк (row_ptr): содержит индексы значений и колонок, указывающие на начало каждой строки.

Таким образом, CSR формат хранения разреженной матрицы позволяет экономить память за счет исключения нулевых значений из хранения и сокращения количества хранимых элементов.

Процесс преобразования матрицы в CSR формат состоит из нескольких шагов:

1. Определение ненулевых элементов матрицы и их значений.
2. Определение индексов столбцов для каждого ненулевого элемента.
3. Определение массива индексов строк, указывающих на начало каждой строки.

Для наглядности рассмотрим пример. Допустим, у нас есть следующая матрица A размером 3×3:

Сначала определим ненулевые элементы матрицы и их значений:

Затем определим индексы столбцов для каждого ненулевого элемента:

Наконец, определим массив индексов строк, указывающих на начало каждой строки:

Теперь мы можем использовать эти три массива для обращения к элементам матрицы, проведения операций сложения и умножения, а также получения доступа к соседним элементам.

Важно заметить, что при работе с разреженными матрицами в CSR формате необходимо учитывать размеры массивов. Например, для матрицы размером MxN массивы values и columns будут содержать M ненулевых элементов, а массив row_ptr будет содержать M+1 элементов, где каждый элемент указывает на начало новой строки. Это означает, что хранение разреженной матрицы в CSR формате может быть выгодно только при наличии большого количества нулевых элементов.

Кроме того, использование CSR формата имеет свои недостатки. Например, обращение к произвольному элементу матрицы может потребовать поиска в массиве значений и массиве колонок, что может замедлить производительность при обработке больших матриц. Также операции модификации, такие как добавление или удаление элементов, могут быть сложными из-за необходимости перестраивать массивы.

В заключение, разреженный строчный формат (CSR) представляет собой эффективный и компактный способ хранения разреженных матриц, особенно в случае большого количества нулевых элементов. Он позволяет сократить использование памяти и обеспечить быстрый доступ к элементам матрицы. Однако использование CSR формата также имеет свои ограничения и

недостатки, которые необходимо учитывать при выборе подходящего формата хранения для конкретного приложения.

Используемое оборудование

ЭВМ iMac 2012 Late 2012 21.5’

Процессор Intel Core i5 4 ядра

ОП 16324 Мб

НМД 524288 Мб

Реализация программы

Структура файла с данными

Перед обсуждением структуры программы по работе с простейшей базой данной, следует рассмотреть формат хранения информации в файлах.

Для этого достаточно будет привести пример содержимого одного из таких файлов. По задумке, данные будут храниться в формате CSV (comma separated values) строка за строкой. Это обеспечит более простое чтение текстовых файлов программой:

Fedor,4,i9+i9+i5+i7,1024,1060,6144,2,2+2,3,Ubuntu

Masha,3,i3+i3+i7,2048,1050,2048,3,8+16+8,4,Windows8

Petya,1,i5,4096,1050,2048,2,16+2,3,Windows7

Fedor,3,i9+i7+i5,6144,1060,4096,3,16+1+32,4,Windows10

Vitaly,4,i7+i9+i3+i9,5120,1060,2048,2,16+16,3,Windows7

Ibragim,2,i7+i7,6144,1050,6144,2,16+16,5,Windows7

Vitaly,1,i3,6144,1080,2048,2,32+3,3,Windows8

То есть каждый отдельный компьютер хранится в отдельной строке. Поэтому, если у студента имеется несколько компьютеров, они всё равно будут записаны в разных строках.

Заметим также, что если компьютер студента включает в себя несколько однотипных элементов, они должны быть разделены символом «+».

Таким образом, чтение файла в дальнейшем будет достигаться посредством токенизации в процессе лексического анализа строки относительно разделителей – запятых и символов «+».

Программа и структуры

vector* create_vector(int volume): Эта функция создает вектор с начальным объемом. Она выделяет память для вектора, инициализирует его размер как 0 и выделяет память для массива данных на основе указанного объема.

void print_vector(vector* v): Эта функция выводит значения заданного вектора на стандартный вывод. Она перебирает массив данных вектора и выводит каждый элемент с одним знаком после запятой.

void add(vector* v, double value): Эта функция добавляет значение в вектор. Если вектор уже полон, он увеличивает его объем на один и перераспределяет память. Затем она добавляет значение в массив данных и обновляет размер вектора.

void free_vector(vector* v): Эта функция освобождает память, выделенную для вектора. Сначала она освобождает массив данных, а затем освобождает саму структуру вектора.

sparse* read_matrix(FILE* fd): Эта функция читает матрицу из файла в плотном представлении и преобразует ее в разреженное представление. Она выделяет память для структуры sparse, считывает размер матрицы и заполняет векторы cip, pi и ye соответствующим образом.

void print_sparse(sparse* sp): Эта функция выводит разреженное представление матрицы. Она выводит векторы cip, pi и ye, используя функцию print_vector.

void transform(sparse* sp): Это функция, специфичная для "Задания по варианту 2". Она преобразует разреженную матрицу, разделяя каждый элемент в строке на максимальное по модулю значение в этой строке.

void print_as_regular(sparse* sp): Эта функция выводит разреженное представление матрицы в виде обычной матрицы. Она перебирает строки и столбцы матрицы и выводит значения в обычном формате.

void free_sparse(sparse* v): Эта функция освобождает память, выделенную для разреженной матрицы. Сначала она освобождает векторы cip, pi и ye, а затем освобождает структуру sparse саму по себе.

Последние 5 функций в полной мере позволяют оперировать классом разреженных матриц представленных в трех векторах.

Распечатка программы

Struct.h

```
#ifndef MATRIX
#define MATRIX

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#define MAX_SIZE 100

// Определение структуры данных вектора
```

```
typedef struct {
    double* data;     // Массив для хранения элементов
    int size;   // Текущий размер вектора
    int volume;
} vector;

typedef struct matrix {
    int rows, cols;
    vector* cip;
    vector* pi;
    vector* ye;
} sparse;

vector* create_vector(int volume);

void print_vector(vector* v);

void add(vector* v, double value);

void free_vector(vector* v);

sparse* read_matrix(FILE* fd);

void print_sparse(sparse* v);

// === Задание по варианту 2 ===
void transform(sparse* sp);
// ============================
```

```
void print_as_regular(sparse* sp);

void free_sparse(sparse* v);



#endif
```

Struct.c

```
#include "struct.h"

#include "struct.h"

// Функция создания вектора
vector* create_vector(int volume) {
    vector* v = (vector*)malloc(sizeof(vector));
    v->size = 0;
    v->volume = volume;
    v->data = calloc(volume, sizeof(double));
    return v;
}

// Функция для вывода значений вектора
void print_vector(vector* v) {
    if (v == NULL) {
        return;
    }
    for (int i = 0; i < v->size; i++) {
        printf("[%.1lf] ", v->data[i]);
```

```
    }
    printf("\n");
    return;
}

// Функция для добавления значения вектору
void add(vector* v, double value) {
    if (v->size >= v->volume) {
        v->volume++;
        v->data = (double*)realloc(v->data, v->volume * sizeof(double));
    }
    v->data[v->size++] = value;
    return;
}

// Функция для освобождения памяти выделенной для вектора
void free_vector(vector* v) {
    free(v->data);
    v->data = NULL;
    free(v);
    return;
}

// Функция для чтения матрицы из файла и представления ее в разреженном виде
sparse* read_matrix(FILE* fd) {
    int rows, cols;
    if (fscanf(fd, "%d %d", &rows, &cols) != 2) {
        return NULL; // не удалось прочитать размер матрицы, предполагаем конец файла
    }
```

```
double matrix[cols][rows];
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        fscanf(fd, "%lf", &matrix[i][j]);
    }
}
sparse* sp = (sparse*)malloc(sizeof(sparse));
sp->rows = rows;
sp->cols = cols;
sp->cip = create_vector(rows);
sp->pi = create_vector(cols * rows + 1);
sp->ye = create_vector(cols * rows);
for (int i = 0; i < rows; i++) {
    bool has_values = false;
    int col_start = sp->pi->size;
    for (int j = 0; j < cols; j++) {
        if (matrix[i][j] == 0.0) {
            continue;
        }
        has_values = true;
        add(sp->ye, matrix[i][j]);
        add(sp->pi, j + 1);
    }
    if (has_values) {
        add(sp->cip, col_start);
    } else {
        add(sp->cip, -1); // если нет значений в строке
    }
}
```

```
    add(sp->pi, 0.0);
    return sp;
}

// Функция для вывода разреженного вида матрицы
void print_sparse(sparse* sp) {
    printf("CIP: ");
    print_vector(sp->cip);
    printf("PI:  ");
    print_vector(sp->pi);
    printf("YE:  ");
    print_vector(sp->ye);
}

// Функция для преобразования разреженного вида матрицы
void transform(sparse* sp) {
    double max = 0;
    int max_col = 0;
    int col = 0;
    int prev_col = 0;
    for (int i = 0; i < sp->ye->size; i++) {
        double value = fabs(sp->ye->data[i]);
        if (value > max) {
            max_col = 0;
        }
        if (value >= max) {
            max = value;
            col = (int)sp->pi->data[i];
            if (col > max_col) {
```

```
            if (max_col == 0) {
                prev_col = col;
            } else {
                prev_col = max_col;
            }
            max_col = col;
        }
    }
}
for (int i = 0; sp->pi->data[i] != 0.0; i++) {
    col = (int)sp->pi->data[i];
    if (col == prev_col) {
        sp->ye->data[i] /= max;
    }
}
return;
}

// Функция для вывода разреженного вида матрицы в виде обычной матрицы
void print_as_regular(sparse* sp) {
    for (int i = 0; i < sp->rows; i++) {
        double row[sp->cols];
        for (int r = 0; r < sp->cols; r++) {
            row[r] = 0.0;
        }
        // Обозначить нижнюю границу для PI в текущей строке
        int l_bound = sp->cip->data[i];
        if (l_bound != -1) {
            int up_idx = i + 1;
```

```
            int r_bound;
            // Обрабатываем условие для последней строки
            if (up_idx >= sp->rows) {
                r_bound = sp->pi->size - 1;
            } else {
                r_bound = sp->cip->data[up_idx];
            }
            // Сдвигаем правую границу индекса если она равна -1
            while (r_bound == -1) {
                up_idx++;
                r_bound = sp->cip->data[up_idx];
            }
            for (int p = l_bound; p < r_bound; p++) {
                int col = sp->pi->data[p];
                row[col - 1] = sp->ye->data[p];
            }
        }
        for (int r = 0; r < sp->cols; r++) {
            printf("|%.1lf| ", row[r]);
        }
        printf("\n");
    }
    return;
}

// Функция для освобождения памяти выделенной под разреженную матрицу
void free_sparse(sparse* v) {
    if (v == NULL) {
        return;
```

```
    }
    free_vector(v->cip);
    free_vector(v->pi);
    free_vector(v->ye);
    free(v);
    return;
}
```

Main.c

```
#include "struct.h"

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

int main(int argc, char **argv) {
    if (argc != 2 || argv[1] == NULL) {
        printf("Использование: lab <файл>\n");
        exit(1);
    }

    FILE* fd = fopen(argv[1], "r");
    if (fd == NULL) {
        perror("Ошибка открытия файла");
        exit(1);
    }

    sparse* sp;
    int n = 1;
```

```
    while (true) {
        sp = read_matrix(fd);
        if (sp == NULL) {
            break;
        }

        printf(">>=================================================\n");
        printf("Матрица №%d в разреженном виде:\n", n);
        print_sparse(sp);

        transform(sp);

        printf("После преобразования:\n");
        print_sparse(sp);

        printf("В обычном виде (преобразованная):\n");
        print_as_regular(sp);

        printf(">>=================================================\n\n");

        free_sparse(sp);
        n++;
    }



    return 0;
}
```

Результат работы программы

test.txt

```
5 5
0 0 2 3 3
5 1 0 0 2
3 5 1 6 0
0 0 1 0 0
0 3 5 2 4

4 4
0 0 0 3
5 1 0 0
3 0 0 6
0 0 1 0

3 3
0 1 0
5 1 0
3 0 0

3 2
3 0
0 1
0 0
```

Матрица подаваемая на вход программе

```
>>================================================
Матрица №1 в разреженном виде:
CIP: [0.0] [3.0] [6.0] [10.0] [11.0]
PI:  [3.0] [4.0] [5.0] [1.0] [2.0] [5.0] [1.0] [2.0] [3.0] [4.0] [3.0] [2.0] [3.0] [4.0] [5.0] [0.0]
YE:  [2.0] [3.0] [3.0] [5.0] [1.0] [2.0] [3.0] [5.0] [1.0] [6.0] [1.0] [3.0] [5.0] [2.0] [4.0]
После преобразования:
CIP: [0.0] [3.0] [6.0] [10.0] [11.0]
PI:  [3.0] [4.0] [5.0] [1.0] [2.0] [5.0] [1.0] [2.0] [3.0] [4.0] [3.0] [2.0] [3.0] [4.0] [5.0] [0.0]
YE:  [2.0] [0.5] [3.0] [5.0] [1.0] [2.0] [3.0] [5.0] [1.0] [1.0] [1.0] [3.0] [5.0] [0.3] [4.0]
В обычном виде (преобразованная):
|0.0| |0.0| |2.0| |0.5| |3.0|
|5.0| |1.0| |0.0| |0.0| |2.0|
|3.0| |5.0| |1.0| |1.0| |0.0|
|0.0| |0.0| |1.0| |0.0| |0.0|
|0.0| |3.0| |5.0| |0.3| |4.0|
>>================================================

>>================================================
Матрица №2 в разреженном виде:
CIP: [0.0] [1.0] [3.0] [5.0]
PI:  [4.0] [1.0] [2.0] [1.0] [4.0] [3.0] [0.0]
YE:  [3.0] [5.0] [1.0] [3.0] [6.0] [1.0]
После преобразования:
CIP: [0.0] [1.0] [3.0] [5.0]
PI:  [4.0] [1.0] [2.0] [1.0] [4.0] [3.0] [0.0]
YE:  [0.5] [5.0] [1.0] [3.0] [1.0] [1.0]
В обычном виде (преобразованная):
|0.0| |0.0| |0.0| |0.5|
|5.0| |1.0| |0.0| |0.0|
|3.0| |0.0| |0.0| |1.0|
|0.0| |0.0| |1.0| |0.0|
>>================================================

>>================================================
Матрица №3 в разреженном виде:
CIP: [0.0] [1.0] [3.0]
PI:  [2.0] [1.0] [2.0] [1.0] [0.0]
YE:  [1.0] [5.0] [1.0] [3.0]
После преобразования:
CIP: [0.0] [1.0] [3.0]
PI:  [2.0] [1.0] [2.0] [1.0] [0.0]
YE:  [1.0] [1.0] [1.0] [0.6]
В обычном виде (преобразованная):
|0.0| |1.0| |0.0|
|1.0| |1.0| |0.0|
|0.6| |0.0| |0.0|
>>================================================

>>================================================
Матрица №4 в разреженном виде:
CIP: [0.0] [1.0] [-1.0]
PI:  [1.0] [2.0] [0.0]
YE:  [3.0] [1.0]
После преобразования:
CIP: [0.0] [1.0] [-1.0]
PI:  [1.0] [2.0] [0.0]
YE:  [1.0] [1.0]
В обычном виде (преобразованная):
|1.0| |0.0|
|0.0| |0.0|
|0.0| |0.0|
>>================================================
```

Вывод

Данная курсовая работа познакомила меня с таким понятием, как разреженные матрицы, с которыми ранее я не сталкивался. Я узнал, что разреженные матрицы представляют собой матрицы, в которых большая часть элементов имеет значение 0, и их хранение в обычном виде может быть неэффективным с точки зрения использования памяти.

Благодаря изучению данной темы, я осознал, что разреженные матрицы можно эффективно хранить в специальном формате, который позволяет экономить память устройства. В моем случае я использовал формат представления разреженных матриц в разреженном строчном формате, хранимых в трех векторах.

Этот подход к хранению разреженных матриц позволяет существенно сократить объем занимаемой памяти, поскольку мы учитывает только ненулевые значения и их позиции.

Теперь я понимаю, что использование разреженных матриц и соответствующего формата их хранения может быть очень полезным в случаях, когда матрицы имеют большой размер и содержат множество нулевых значений. Это также позволяет повысить эффективность работы с матрицами, особенно при выполнении операций над ними.

Полученные знания и навыки в области разреженных матриц открывают для меня новые возможности в разработке и оптимизации алгоритмов, где использование матриц играет важную роль. Данная тема имеет широкое применение в различных областях, таких как машинное обучение, графические вычисления, численные методы и другие, где работа с большими объемами данных и оптимизация ресурсов являются важными аспектами.

Использованные источники

1. https://ru.wikipedia.org/wiki/Разреженная_матрица
2. https://qaa-engineer.ru/hranenie-razrezhennoj-matriczy-v-razrezhennom-strochnom-formate-csr-ili-crs/