МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Институт№8: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**ПРАКТИКУМ**

По циклу дисциплин «Информатика»

2 семестр

Тема:

«Разреженные матрицы»

|  |  |
| --- | --- |
| Группа: | М8О-106Б-22 |
| Студент: | Абдисаламов Э. |
| Преподаватель: | Дубинин А. В. |
| Оценка: |  |
| Дата: |  |

Москва, 2023

Содержание

[Введение 4](#_Toc131537686)

[1. Теоретическая часть 5](#_Toc131537687)

[1.1 Матрица 5](#_Toc131537688)

[1.2 Разреженные матрицы 6](#_Toc131537696)

[1.3 Вектор как структура данных 8](#_Toc131537697)

[2. Практическая часть 9](#_Toc131537698)

[2.1 Задание 9](#_Toc131537699)

[2.2. Вариант схемы размещения матрицы 10](#_Toc131537702)

[2.3 Вариант преобразований: 10](#_Toc131537703)

[2.4 Использованные переменные и функции 10](#_Toc131537704)

[2.5 Протокол 11](#_Toc131537706)

[2.6 Оценка сложности функций 12](#_Toc131537707)

[2.7 Входные и выходные данные. Тесты 12](#_Toc131537708)

[Заключение 17](#_Toc131537709)

[Список использованных источников 18](#_Toc131537710)

# Введение

Разреженные матрицы — это специальный тип матриц, в котором большинство элементов равны нулю. Они широко используются в различных областях, таких как теория графов, компьютерная графика, физика и другие науки.

Они могут быть представлены в виде массивов, списков, деревьев и других структур данных. Одна из особенностей разреженных матриц заключается в том, что выбор оптимального алгоритма для выполнения операций с этой матрицей зависит от типа матрицы и ее представления.

Разреженные матрицы также используются в области обработки больших данных, так как они позволяют представлять данные более компактно и эффективно. Например, в задачах машинного обучения, где может быть необходимо работать с матрицами больших размеров, использование разреженных матриц может значительно ускорить процесс обучения и уменьшить потребление памяти.

Таким образом, разреженные матрицы имеют широкое применение в различных областях науки и техники, и знание алгоритмов работы с ними может быть полезно для решения различных задач, связанных с обработкой больших данных и машинным обучением.

# 1. Теоретическая часть

## 1.1 Матрица

## Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы. Матрицу можно также представить в виде функции двух дискретных аргументов. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

## Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

## Для матрицы определены следующие алгебраические операции:

## сложение матриц, имеющих один и тот же размер;

## умножение матриц подходящего размера (матрицу, имеющую n столбцов, можно умножить справа на матрицу, имеющую n строк);

## в том числе умножение матрицы на вектор-столбец и умножение вектор-строки на матрицу (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);

## умножение матрицы на элемент основного кольца или поля (то есть скаляр).

## 1.2 Разреженные матрицы

Разрежённая матрица — матрица с преимущественно нулевыми элементами. В противном случае, если бо́льшая часть элементов матрицы ненулевые, матрица считается плотной.

Среди специалистов нет единства в определении того, какое именно количество ненулевых элементов делает матрицу разрежённой. Разные авторы предлагают различные варианты. Для матрицы порядка n число ненулевых элементов:

* есть O(n). Такое определение подходит разве что для теоретического анализа асимптотических свойств матричных алгоритмов;
* в каждой строке не превышает 10 в типичном случае;
* ограничено
* таково, что для данного алгоритма и вычислительной системы имеет смысл извлекать выгоду из наличия в ней нулей.

Огромные разрежённые матрицы часто возникают при решении таких задач, как дифференциальное уравнение в частных производных.

При хранении и преобразовании разрежённых матриц в компьютере бывает полезно, а часто и необходимо, использовать специальные алгоритмы и структуры данных, которые учитывают разрежённую структуру матрицы. Операции и алгоритмы, применяемые для работы с обычными, плотными матрицами, применительно к большим разрежённым матрицам работают относительно медленно и требуют значительных объёмов памяти. Однако разрежённые матрицы могут быть легко сжаты путём записи только своих ненулевых элементов, что снижает требования к компьютерной памяти.

Представление

Представление разреженной матрицы в виде 2D-массива приводит к потере большого объема памяти, поскольку нули в матрице в большинстве случаев бесполезны. Итак, вместо хранения нулей с ненулевыми элементами мы храним только ненулевые элементы. Это означает хранение ненулевых элементов с **тройками - (строка, столбец, значение).**

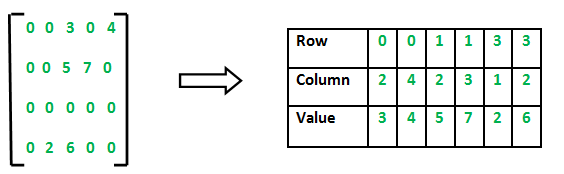
Представления разреженной матрицы могут быть выполнены многими способами ниже приведены два распространенных представления:

1. Представление массива
2. Представление связанного списка

**Способ 1: использование массивов:**

2D-массив используется для представления разреженной матрицы, в которой есть три строки, названные как

* **Строка:**индекс строки, в которой расположен ненулевой элемент
* **Столбец:**индекс столбца, в котором расположен ненулевой элемент
* **Значение:**значение ненулевого элемента, расположенного в индексе – (строка, столбец)

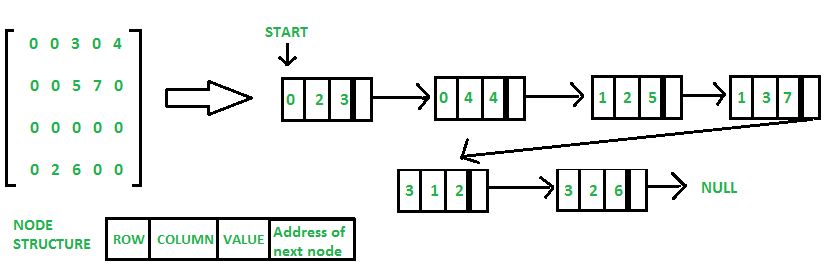


Временная сложность: O(NM), где N - количество строк в разреженной матрице, а M - количество столбцов в разреженной матрице.

Вспомогательное пространство: O(NM), где N - количество строк в разреженной матрице, а M - количество столбцов в разреженной матрице.

**Способ 2: использование связанных списков**  
В связанном списке каждый узел имеет четыре поля. Эти четыре поля определяются как:

* **Строка:**индекс строки, в которой расположен ненулевой элемент
* **Столбец:**индекс столбца, в котором расположен ненулевой элемент
* **Значение:**значение ненулевого элемента, расположенного в индексе – (строка, столбец)
* **Следующий узел:**адрес следующего узла



Временная сложность: O(N\* M), где N - количество строк в разреженной матрице, а M - количество столбцов в разреженной матрице.

Вспомогательное пространство: O(1)

1.3 Вектор как структура данных

Вектор (одномерный динамический массив) - структура данных с элементами одного и того же типа типа. Каждый элемент вектора имеет уникальный в рамках заданного вектора номер. Обращение к элементу вектора выполняется по имени вектора и номеру требуемого элемента.

Машинное представление вектора. Адресация элементов структур.

Элементы вектора размещаются в памяти в подряд расположенных ячейках памяти. Под элемент вектора выделяется количество байт памяти, определяемое базовым типом элемента этого вектора. Необходимое число байтов памяти для хранения одного элемента вектора называется слотом. Размер памяти для хранения вектора определяется произведением длины слота на число элементов.

2. Практическая часть

2.1 Задание

Составить программу на языке Си с процедурами и/или функциями для обработки прямоугольных разреженных матриц с элементами целого (группы 6, 8), вещественного (группы 2-5), или комплексного (группы 1, 7) типов, которая:

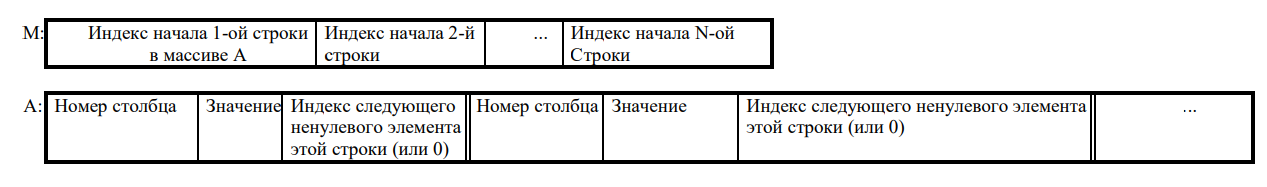
1. вводит матрицы различного размера, представленные во входном текстовом файле в обычном формате (по строкам), с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой;
2. печатает введенные матрицы во внутреннем представлении согласно заданной схеме размещения и в обычном (естественном) виде;
3. выполняет необходимые преобразования разреженных матриц (или вычисления над ними) путем обращения к соответствующим процедурам и/или функциям;
4. печатает результат преобразования (вычисления) согласно заданной схеме размещения и в обычном виде.

В процедурах и функциях предусмотреть проверки и печать сообщений в случаях ошибок в задании параметров. Для отладки использовать матрицы, содержащие 5–10% ненулевых элементов с максимальным числом элементов 100.

2.2. Вариант схемы размещения матрицы

Все матрицы m \* n хранятся по строкам, в порядке возрастания индексов ненулевых элементов.

1. Цепочка ненулевых элементов в векторе A со строчным индексированием (индексы в массиве M равны 0, если соответствующая строка матрицы содержит только нули):



Индекс, равный нулю, означает отсутствие ненулевых элементов в строке (или в ее остатке). Если матрицы не изменяются программой, возможна экономия памяти за счет отказа от хранения в массиве А индексов следующего элемента столбца (когда элементы идут подряд).

2.3 Вариант преобразований:

2. Определить максимальный по модулю элемент матрицы и разделить на него все элементы столбца, в котором он находится. Если таких элементов несколько, обработать предпоследний столбец, содержащий такой элемент.

2.4 Использованные переменные и функции

* В структуре matrix:

| **Тип** | **Имя переменной** | **Описание** |
| --- | --- | --- |
| int | n | Количество строк |
| int | m | Количество столбцов |
| int\_vector | row | Массив М |
| mx\_vector | elems | Массив А |
| dbl\_mx\_el | max\_el | Максимальный элемент матрицы |
| int | prev\_clmn\_max\_el | Индекс предыдущего |

* В структуре dbl\_mx\_el:

| **Тип переменной** | **Имя переменной** | **Описание** |
| --- | --- | --- |
| int | clmn | Индекс столбца |
| double | val | Значение |
| int | next\_el\_idx | Индекс следующего элемента в этой строке |

* В функции read\_mtx:

|  |  |  |
| --- | --- | --- |
| Тип переменной | Имя | Описание |
| int | n | Количество строк |
| int | m | Количество столбцов |
| matrix | mx | Создаваемая матрица |
| bool | row\_empty | Индикатор пустоты строки |
| double | num | Читаемое значение |
| dbl\_mx\_el | prev\_el | Предыдущий элемент в этой строке |
| dbl\_mx\_el | el | Текущий элемент |

* В функции print\_mtx:

|  |  |  |
| --- | --- | --- |
| Тип переменной | Имя | Описание |
| int | elem\_idx | Индекс в массиве А текущего элемента |
| dbl\_mx\_el | elem | Текущий элемент |

* В функции kp7\_var2:

|  |  |  |
| --- | --- | --- |
| Тип переменной | Имя | Описание |
| int | elem\_idx | Индекс в массиве А текущего элемента |
| dbl\_mx\_el | elem | Текущий элемент |
| dbl\_mx\_el | changeable\_clmn | Изменяемый столбец |
| dbl\_mx\_el | new\_el | Измененный элемент |

## 2.5 Протокол

* Kp7.h:

#pragma once

#include <stdbool.h>

#include "vector/vector\_dbl.h"

#include "vector/vector\_int.h"

#include "vector/vector\_mx.h"

typedef struct {

    int n, m;

    int\_vector row;

    mx\_vector elems;

} matrix;

typedef struct {

    double val;

    int prev\_clmn\_max\_el;

    int clmn;

} max\_el\_mx;

matrix read\_mtx();

void print\_mtx\_user(matrix \*mx);

void print\_mtx\_private(matrix \*mx);

void kp7\_var2(matrix \*mx);

max\_el\_mx get\_max\_el(matrix \*mx);

* kp7.c:

#include <stdbool.h>

#include <math.h>

#include <stdio.h>

#include "kp7.h"

max\_el\_mx get\_max\_el(matrix \*mx){

    max\_el\_mx max\_el;

    max\_el.clmn = 0;

    max\_el.row = 0;

    max\_el.val = 0;

    max\_el.prev\_clmn\_max\_el = -1;

    int cur\_row = 0;

    for (int i = 0; i < mx->elems.size; i++){

        dbl\_mx\_el elem = mx\_get\_el(&mx->elems, i);

        if ((fabs(elem.val) == fabs(max\_el.val)) && (max\_el.row == cur\_row)){

            max\_el.prev\_clmn\_max\_el = max\_el.clmn;

            printf("biba - %d\n", elem.clmn);

        }

        if (fabs(elem.val) >= fabs(max\_el.val)) {

            max\_el.val = elem.val;

            max\_el.clmn = elem.clmn;

            max\_el.row = cur\_row;

        }

        if (elem.next\_el\_idx == 0) cur\_row++;

    }

    if (max\_el.val == 0) max\_el.val = -1;

    return max\_el;

}

matrix read\_mtx(){

    int n, m;

    scanf("%d %d", &n, &m);

    matrix mx;

    mx.n = n;

    mx.m = m;

    int\_init(&mx.row);

    mx\_init(&mx.elems);

    for (int i = 0; i < n; i++){

        bool row\_empty = true;

        for (int j = 0; j < m; j++){

            double num;

            scanf("%lf", &num);

            if (num != 0){

                if (row\_empty){

                    row\_empty = false;

                    int\_push\_back(&mx.row, mx\_get\_size(&mx.elems));

                } else {

                    dbl\_mx\_el prev\_el = mx\_pop\_back(&mx.elems);

                    prev\_el.next\_el\_idx = mx\_get\_size(&mx.elems) + 1;

                    mx\_push\_back(&mx.elems, prev\_el);

                }

                dbl\_mx\_el el;

                el.clmn = j;

                el.val = num;

                el.next\_el\_idx = 0;

                mx\_push\_back(&mx.elems, el);

            }

        }

        if (row\_empty){

            int\_push\_back(&mx.row, -1);

        }

    }

    return mx;

}

void print\_mtx\_user(matrix \*mx){

    printf("Matrix:\n");

    for (int i = 0; i < mx->n; i++){

        bool row\_empty = false;

        int elem\_idx = int\_get\_el(&mx->row, i);

        if (elem\_idx == -1) row\_empty = true;

        dbl\_mx\_el elem = mx\_get\_el(&mx->elems, elem\_idx);

        for (int j = 0; j < mx->m; j++){

            if (!row\_empty && elem.clmn == j){

                printf("%lf ", elem.val);

                elem\_idx = elem.next\_el\_idx;

                if (elem\_idx == 0) row\_empty = true;

                elem = mx\_get\_el(&mx->elems, elem\_idx);

            } else {

                printf("0.000000 ");

            }

        }

        printf("\n");

    }

}

void print\_mtx\_private(matrix \*mx){

    printf("Matrix M: | ");

    for (int i = 0; i < mx->row.size; i++){

        printf("%d | ", int\_get\_el(&mx->row, i));

    }

    printf("\nMatrix A: ");

    for (int i = 0; i < mx->elems.size; i++){

        dbl\_mx\_el elem = mx\_get\_el(&mx->elems, i);

        printf("[ %d | %lf | %d ] ", elem.clmn, elem.val, elem.next\_el\_idx);

    }

    printf("\n");

}

void kp7\_var2(matrix \*mx){

    max\_el\_mx max\_el = get\_max\_el(mx);

    int changeable\_clmn;

    printf("\n%d\n", max\_el.prev\_clmn\_max\_el);

    if (max\_el.prev\_clmn\_max\_el == -1){

        changeable\_clmn = max\_el.clmn;

    } else {

        changeable\_clmn = max\_el.prev\_clmn\_max\_el;

    }

    for (int i = 0; i < mx->elems.size; i++){

        dbl\_mx\_el elem = mx\_get\_el(&(mx->elems), i);

        if (elem.clmn == changeable\_clmn){

            elem.val = (elem.val)/max\_el.val;

            mx\_set\_el(&(mx->elems), i, elem);

        }

    }

}

* main.c:

#include <stdbool.h>

#include <stdio.h>

#include "kp7.h"

int main(){

    matrix mx = read\_mtx();

    kp7\_var2(&mx);

    print\_mtx\_user(&mx);

    print\_mtx\_private(&mx);

    return 0;

}

2.6 Оценка сложности функций

k – кол-во ненулевых элементов,

n – кол-во строк,

m – кол-во столбцов

get\_max\_el – O(k)

read\_mtx – O(n\*m)

print\_mtx\_user – O(n\*m)

print\_mtx\_private – O(k+n)

kp7\_var2 – O(k)

2.7 Входные и выходные данные. Тесты

quibex@STARK:~/labs/kp7$ ./kp7.out << end

> 5 5

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

> end

-1

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

Array M: | -1 | -1 | -1 | -1 | -1 |

Array A:

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 5

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

end

-1

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 1.000000

Array M: | -1 | -1 | -1 | -1 | 0 |

Array A: [ 4 | 1.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 5

0 0 0 0 0

0 0 0 14 0

0 12 0 0 0

0 0 0 0 0

0 0 0 10 -15

end

-1

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 14.000000 0.000000

0.000000 12.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 10.000000 1.000000

Array M: | -1 | 0 | 1 | -1 | 2 |

Array A: [ 3 | 14.000000 | 0 ] [ 1 | 12.000000 | 0 ] [ 3 | 10.000000 | 3 ] [ 4 | 1.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 5

0 0 0 0 0

0 0 0 14 0

0 12 0 0 0

0 0 0 0 0

0 0 0 15 -15

end

biba - 4

3

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 -0.933333 0.000000

0.000000 12.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 -1.000000 -15.000000

Array M: | -1 | 0 | 1 | -1 | 2 |

Array A: [ 3 | -0.933333 | 0 ] [ 1 | 12.000000 | 0 ] [ 3 | -1.000000 | 3 ] [ 4 | -15.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 5

0 0 0 0 0

0 0 0 14 0

0 12 0 0 0

0 0 0 0 0

0 0 -15 15 -15

end

biba - 3

biba - 4

3

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 -0.933333 0.000000

0.000000 12.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 -15.000000 -1.000000 -15.000000

Array M: | -1 | 0 | 1 | -1 | 2 |

Array A: [ 3 | -0.933333 | 0 ] [ 1 | 12.000000 | 0 ] [ 2 | -15.000000 | 3 ] [ 3 | -1.000000 | 4 ] [ 4 | -15.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 5

0 0 0 0 0

0 0 0 14 0

0 12 0 0 0

0 0 0 0 0

0 0 -15 13 -15

end

biba - 4

2

Matrix:

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 14.000000 0.000000

0.000000 12.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 1.000000 13.000000 -15.000000

Array M: | -1 | 0 | 1 | -1 | 2 |

Array A: [ 3 | 14.000000 | 0 ] [ 1 | 12.000000 | 0 ] [ 2 | 1.000000 | 3 ] [ 3 | 13.000000 | 4 ] [ 4 | -15.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 3

0 0 0

0 0 0

0 0 0

0 0 4

0 0 12

end

-1

Matrix:

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.000000

0.000000 0.000000 0.333333

0.000000 0.000000 1.000000

Array M: | -1 | -1 | -1 | 0 | 1 |

Array A: [ 2 | 0.333333 | 0 ] [ 2 | 1.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

5 3

0 0 0

0 5 0

0 0 0

3 0 4

0 0 12

end

-1

Matrix:

0.000000 0.000000 0.000000

0.000000 5.000000 0.000000

0.000000 0.000000 0.000000

3.000000 0.000000 0.333333

0.000000 0.000000 1.000000

Array M: | -1 | 0 | -1 | 1 | 3 |

Array A: [ 1 | 5.000000 | 0 ] [ 0 | 3.000000 | 2 ] [ 2 | 0.333333 | 0 ] [ 2 | 1.000000 | 0 ]

quibex@STARK:~/labs/kp7$ ./kp7.out << end

3 5

0 0 0 3 4

3 0 4 14 17

0 0 12 -18 1

end

-1

Matrix:

0.000000 0.000000 0.000000 -0.166667 4.000000

3.000000 0.000000 4.000000 -0.777778 17.000000

0.000000 0.000000 12.000000 1.000000 1.000000

Array M: | 0 | 2 | 6 |

Array A: [ 3 | -0.166667 | 1 ] [ 4 | 4.000000 | 0 ] [ 0 | 3.000000 | 3 ] [ 2 | 4.000000 | 4 ] [ 3 | -0.777778 | 5 ] [ 4 | 17.000000 | 0 ] [ 2 | 12.000000 | 7 ] [ 3 | 1.000000 | 8 ] [ 4 | 1.000000 | 0 ]

# Заключение

В результате реализации разреженных матриц на языке С, мы можем эффективно использовать память и ускорить вычисления, особенно для больших матриц. Разреженные матрицы также удобны для работы с разреженными данными, такими как изображения или тексты, где большинство ячеек матрицы содержат нули.

Однако, реализация разреженных матриц может быть сложной и требует дополнительного кода для обработки специальных операций, таких как умножение матриц или транспонирование. Также, при работе с разреженными матрицами может возникнуть необходимость в конвертации матрицы в обычную (плотную) форму, что может привести к дополнительным затратам по памяти.

В целом, реализация разреженных матриц на языке С имеет свои плюсы и минусы, и может быть полезна в зависимости от конкретной задачи, которую необходимо решить.

# Список использованных источников

1. Беллман Р. Введение в теорию матриц.
2. Sparse Matrix and its representations [URL]: https://www.geeksforgeeks.org/sparse-matrix-representation/
3. Вектор как структура данных [URL]: https://intellect.icu/vektor-kak-struktura-dannykh-i-predstavlenie-khranenie-ego-v-pamyati-8510
4. Reginald P. Tewarson. Sparse Matrices. — Academic Press, 1973. — 160 с. — ISBN 0126856508. перевод: Тьюарсон Р. Разрежённые матрицы = Sparse Matrices. — М.: Мир, 1977. — 191 с.