Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ФУНДАМЕНТАЛЬНЫЕ НАУКИ»

КАФЕДРА «ПРИКЛАДНАЯ МАТЕМАТИКА»

Лабораторная работа № 4

по дисциплине «Типы и структуры данных»

Тема Схемы сжатого хранения разреженных матриц

Студент Лямин И.С.

Группа ФН12-31Б

Преподаватели Волкова Л.Л.

Москва, 2024

Содержание

ВВЕДЕНИЕ

Цель работы — реализовать алгоритмы сжатия матриц по схеме Дженнингса и Ренбольдта-Местеньи, алгоритмы операций умножения и сложения между матрицами.

Для достижения цели необходимо выполнить следующие задачи:

1) разобрать суть основных понятий используемых для преобразовании и вычислении
2) разработать алгоритмы упаковки и распаковки матриц по схеме Дженинкса и Рейнбольдта-Местеньи
3) разработать алгоритмы сложения и умножения упакованных матриц
4) реализовать алгоритмы
5) провести тестирование, проверить работоспособность реализаций алгоритмов

1 Аналитическая часть

1.1 Схема Дженнингса

Схема Дженнингса – это схема для сжатого хранения квадратных симметричных матриц. Наиболее эффективно это схема работает с матрицами, элементы которых сгруппированы около главной диагонали. Сжатая матрица представляется в виде двух массивов (AN, D). AN – массив для хранения значимых элементов, а D – массив для хранения координат элементов главной строки относительно массива AN. Значимыми элементами называем все элементы главной диагонали, все ненулевые элементы, расположенные в верхем треугольнике матрицы, и все такие нули, после которых, при просмотре вдоль строки, находятся ненулевые элементы.

Таким образом, вместо хранения двумерного массива из n^2 элементов, будет храниться два массива, первый из m элементов, где m – количество значимых элементов, и второй массив длины n.

Проблема этой схемы это не эффективное хранение матриц с элементами на побочной диагонали.

1.2 Кольцевая схема Рейнбольдта-Местеньи

Схема Рейнбольдта-Местеньи – это схема для сжатого хранения произвольных матриц. Сжатая матрица представляется в виде пяти массивов (AN, NR, NC, JR, JC). AN – массив для хранения значимых элементов, JR – массив для хранения координат, относительно массива AN, элементов которые являются первыми в соответствующей строке, JC – массив для хранения координат, относительно массива AN, элементов которые являются первыми в соответствующем столбце. NC – массив для хранения индексов, в массиве AN, следующего элемента в с столбце изначальной матрицы. NR – массив для хранения индексов, в массиве AN, следующего элемента в с строке изначальной матрицы. Значимыми элементами называем все ненулевые элементы. Для такой схемы хранения, меньшее количество элементов изначальной матрицы являются значимыми, что гарантирует одинаковое количество занимаемой памяти, что диагональной матрицей, что матрицей с элементами на побочной диагонали, а это и есть проблема сжатия по Дженнингсу.

2 Конструкторская часть

2.1 Вспомогательные структуры

Для программной реализации алгоритма сначала опишем вспомогательные структуры

1) `Matrix`
   - Поля: int rows, int columns, int un_zero, int** array,
   - Инициализация: Matrix(), Matrix(int size), Matrix(int rows, int columns),
   - Методы: print() – вывод содержмого,
2) `CompressedMatrix`,
   - Поля: int* AN, int* D, int len_AN = 0, int size,
   - Инициализация: CompressedMatrix(), CompressedMatrix(int* AN, int* D, int len_AN, int size) Matrix(int size),
   - Методы: print() – вывод содержмого,
3) `KRMCompressedMatrix`
   - Поля: int* AN, int* NR, int* NC, int* JR, int* JC, int len_AN, int rows, int columns,
   - Инициализация: KRMCompressedMatrix(), KRMCompressedMatrix(Matrix* mat), KRMCompressedMatrix(map<pair<int, int>, int>* coords, int rows, int columns), KRMCompressedMatrix(int len_AN, int rows, int columns, int*& AN, int*& NR, int*& NC, int*& JR, int*& JC),
   - Методы: print() – вывод содержмого,

Так же во всех структурах переопределён оператор присваивания, учитывающий необходимость чистки памяти из под динамических массивов. Пример на листинге 2.1.

Листинг 2.1 — пример опертара присваивания для структуры CompressedMatrix

```
CompressedMatrix& operator= (const CompressedMatrix& rhs) {
  if (this == (&rhs)) {
    cout << "selfassignment" << "\n";
    return *this;
  }
  clear();

  this->size = rhs.size;
  this->len_AN = rhs.len_AN;
  this->AN = new int[len_AN];
  this->D = new int[size];

  for (int i = 0; i < len_AN; i++) {
    AN[i] = rhs.AN[i];
  }
  for (int i = 0; i < size; i++) {
```

```
      D[i] = rhs.D[i];
    }
    return *this;
  }

  void clear() {
    if (AN != nullptr) {
      delete[] AN;
      AN = nullptr;
    }
    if (D != nullptr) {
      delete[] D;
      D = nullptr;
    }
  }
```

2.2 Вспомогательные функции

Для программной реализации алгоритма опишем вспомогательные функции

1) `usual_addition(Matrix* _1, Matrix* _2)`,
   - — тип возвращаемого значения: Matrix,
   - — описание: суммирует матрицы стандартным алгоритмом,
2) `usual_multiplication(Matrix* array_l, Matrix* array_r)`
   - — тип возвращаемого значения: Matrix,
   - — описание: умножает матрицы стандартным алгоритмом,
3) `parse(string str, int len, Matrix* mat)`
   - — тип возвращаемого значения: int*,
   - — описание: разбивает строку из файла по элементам и проверяет их количество,
4) `read_matrix(Matrix filename)`
   - — тип возвращаемого значения: Matrix,
   - — описание: используя функцию $parse$ для каждой строки из файла, считывает весь файл занося его в массив созданной структуры Matrix, в случае ошибки завершает работу программы,
5) `COMPARE(Matrix* _1, Matrix* _2)`
   - — тип возвращаемого значения: void,
   - — описание: сравнивает по координатно две матрицы и в случае несовпадения элементов, выводит координату по которой элементы различны,

2.3 Алгоритм упаковки матрицы по Дженнингсу

Алгоритм реализован в виде функции create_AN, принимающий в качестве аргументов следующие объекты

1) `matrix` – двумерный массив структуры `Matrix`, которая будет сжата,

2) `size` – размер матрицы,

3) `*an_size` – ссылка на переменную, хранящую длину массива `AN` со значимыми элементами,

4) `**D` – ссылка на динамический массив `D` с индексами элементов главной строки относительно массива `AN`,

5) `**AN` – ссылка на динамический массив `AN` со значимыми элементами,

Поля и начальная настройка

1) `an_loc`
   - — тип: `vector<int>`,
   - — описание: динамический массив, заполняющийся всеми значимыми элементами. После выполнения алгоритма будет скопирован в переданный ,

2) `glob_ind`
   - — тип: `int`,
   - — описание: отображает текущую длину, `an_loc`,

3) `last_ind`
   - — тип: `int`,
   - — описание: количество элементов, определяемое для каждой строки, которе нужно сохранить в `AN`,

Этапы работы алгоритма

Алгоритм реализован в функции `create_an`. `i` – индекс строки матрицы, `j` – индекс столбца матрицы

1) `D[i] = glob_ind`, last_ind = i,

2) Цикл по столбцам матрицы (j от 0 до `size`),
   - — если элемент матрицы с координатами (im j) не ноль, то присваиваем `last_ind` значение `j`,

3) Цикл по столбцам матрицы (j от i до `last_ind + 1`),
   - — добавляем в `an_loc` элемент матрицы с координатами (i, j),

4) Прибавляем к `glob_ind` количество добавленных элементов $last\_ind - i + 1$,

5) цикл продолжается до того как не будут перебраны все строки матрицы,

2.4 Алгоритм распаковки матрицы сжатой по Дженнингсу

Для распаковки матрицы используется функция `matrix_unpackagin`, получающая на вход ссылку на упакованную матрицу `*comp_mat` и возвращающая распакованную матрицу. Алгоритм реализованный в этой функции будет следующий.

Поля и начальная настройка

1) `ans(comp_mat->size)`
   - — тип: `Matrix`,
   - — описание: результирующая матрица,
2) `row`
   - — тип: `int`,
   - — описание: строка результирующей матрицы,
3) `column`
   - — тип: `int`,
   - — описание: столбец результирующей матрицы,

Этапы работы алгоритма

1) Цикл (i от 0 до `comp_mat->len_AN`),
   - — `columnn++`,
   - — если индекс элемента из `AN` (`i`) совпадает с индексом элемента главной диагонали строки `row`, то `row++` и `columnn = row`,
   - — присваиваем значение массива `AN` под индексом `i` элементам матрицы с координами `(row, column)` и `(column, row)`,
2) после окончания цикла, функция возвращает `ans`,

2.5 Алгоритм сложения матриц сжатых по схеме Дженнингса

Для вычисления суммы матриц сжатых по Дженнингсу реализована функция `sum_pack_matrix`, получающая на вход две ссылки на упакованные матрицы `*comp_mat_1` и `*comp_mat_2` и возвращающая упакованную матрицу, которая является суммой двух изначальных. Алгоритм реализованный в этой функции будет следующий.

Поля и начальная настройка

1) `loc_sum`,
   - — тип: `*vector<int>[_1->size]`,
   - — описание:двумерный массив хранящий массивы покоординатно сложенных элементов из первой и второй матриц,

2) result_AN,
   — тип: vector<int>,
   — описание: результирующий векотор, который будет передан для инициализации итоговой структуры,
3) result_D
   — тип: vector<int>,
   — описание: результирующий векотор, который будет передан для инициализации итоговой структуры,
4) len_row_1
   — тип: int,
   — описание: количество значимых элементов в строке первой матрицы,
5) len_row_2
   — тип: int,
   — описание: количество значимых элементов в строке второй матрицы,
6) iter_row
   — тип: int,
   — описание: строка результирующей матрицы,
7) ans
   — тип: CompressedMatrix,
   — описание: итоговая структура,
8) last_index
   — тип: int,
   — описание: количество значимых элементов в строке для результирующей матрицы,

Этапы работы алгоритма

1) цикл while (iter_row < (_1->size - 1)),
   — len_row_1 = _1->D[iter_row + 1] - _1->D[iter_row],
   — len_row_2 = _2->D[iter_row + 1] - _2->D[iter_row],
   — цикл (i от 0 до max(len_row_1, len_row_2)),
      – если (i < len_row_1) то прибавляем элемент из первой матрицы,
      – если (i < len_row_2) то прибавляем элемент из второй матрицы,
   — iter_row++,
2) цикл (i от 0 до _1->size)
   — добавляем в result_D элемент со значением текущей длины массива result_AN,
   — цикл (el от 0 до loc_sum[i].size()),
      – если loc_sum[i][el] не ноль, то обновляем last_ind на el,
   — цикл (j от 0 до last_ind),

– добавляем элемент из `loc_sum` с координатами (i, j) в массив `result_AN`,

3) инициализируем `ans` используя полученные значения,

4) функция возвращает ans,

2.6 Алгоритм упаковки матрицы по схеме Рейнбольдта-Местеньи

Для хранения матрицы используется 5 массивов `AN` – массив с ненулевыми элементами, `NR` –массив в котором на i-ом месте стоит индекс в `AN` элемента являющегося следующим в строке для i-го из `AN`, `NC` –массив в котором на i-ом месте стоит индекс в `AN` элемента являющегося следующим для i-го в столбце, `JR` – массив хранящий на i-ом месте индекс элемента из `AN`, являющегося первым в i-ой строке, `JC` – массив хранящий на i-ом месте индекс элемента из `AN`, являющегося первым в i-ом столбце. Изначально массивы с вхождениями заполнены значениями -1. Обозначим `iter` – текущее количество рассмотренных ненулевых элементов, `Cur_R` – индекс текущего элемента строки, изначально равного `JR[i]`, `Cur_C` – индекс текущего элемента столбца, изначально равного `JC[j]`. Алгоритм следующий.

Этапы работы алгоритма

1) цикл перебирает элементы матрицы слева направо, сверху вниз, координаты [i, j],
   - `NC[iter]` и `NR[iter]` присваиваем значение iter,
   - если JR[i] пустой, то присваиваем ему значение `iter`, тоже самое для JC[j],
   - цикл (пока `Cur_R` < `iter`),
     - `NR[cur_R] = cur_R + 1, cur_R += 1`,
   - закольцовывание ссылок на следующие элементы путём присваивания `NR[iter]` значения индекса первого элемента из строки,
   - цикл (пока `Cur_C` < `iter`),
     - если элемент под индексом, `Cur_C` ссылается на первый элемент в строке, то меняем его ссылку на элемент под индексом `iter` и меняем текущий присваиваем `Cur_C` значение `iter`,
     - иначе, если ссылка на следующий элемент после элемента индексом `Cur_C` не совпадает с индексом `iter`, то меняем индексы переходя к следующему элементу,
   - закольцовывание ссылок на следующие элементы путём присваивания `NC[iter]` значения индекса первого элемента из столбца,

2.7 Алгоритм распаковки матрицы из КРМ

Для распаковки матрицы используются две дополнительные функции `get_IA` и `get_JA`. Эти функции принимают на вход сжатую матрицу и возвращают массив с номерами строк в

которых расположены элементы и номерами столбцов соответственно. Изначально результирующий массив `IA` инициализируется длинной `len_AN` и заполняется значениями -1.

Этапы работы алгоритма

1) цикл перебирает элементы массива `AN` по индексам i,
   — если это первый элемент строки, то присваиваем соответствующему значению в `IA` номер строки,
   — иначе, присваиваем соответствующему значению в `IA` и всем остальным полученным переходом по массиву `NC` значение номера строки,

Полностью аналогичный алгоритм для функции `get_JA`. Для получения распакованной матрицы, применяем обе эти функции и получаем словарь с координатами ненулевых элементов, значения которого координатно присваиваем элементам матрицы.

2.8 Алгоритм сложения матриц в КРМ

Этапы работы алгоритма

1) проверка на совместимость матриц по размерам,
2) получение массивов IA и JA,
3) создание словарь типа `map<pair<int, int>, int>`, который хранит значение элементов новой матрицы по координатам,
4) цикл по элементам первой матрицы,
   — если элемент с такими координатами уже есть в словаре, то прибавляем значение этого элемента соответствующему элементу из словаря,
   — иначе, создается элемент в словаре со значением элемента из матрицы,
5) аналогичный цикл для второй матрицы,
6) проверка на хранение нулей,
7) упаковка словаря в КРМ (аналогично упаковке матрицы см. 2.6),

2.9 Алгоритм умножения матриц в КРМ

Этапы работы алгоритма

1) проверка на совместимость матриц по размерам,
2) получение массивов IA и JA,
3) создание словарей с координатами элементов `coords_1` и `coords_2`,
4) определение размера результирующей матрицы,
5) перебор слева направо, сверху вниз, всех координат новой матрицы,
   — цикл (k от 0 до (количество столбцов новой матрицы)),
      – если есть ключ (i, k) в перовом словаре и есть ключ (k, j) во втором

словаре, то в новый словарь заносится их произведение,

— инициализация сжатой матрицы по полученному словарю,

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации алгоритма использовалась среда разработки Visual Studio 2022, язык программирования, на котором была выполнена реализации алгоритмов — C++. Для компиляции кода использовался компилятор MSVC. Исследование проводилось на ноутбуке (64–разрядная операционная система, процессор x64, частота процессора 3.1 ГГц, модель процессора 12th Gen Intel(R) Core(TM) i5-12500H, оперативная память 16 ГБ)

3.2 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9представлена программная реализация описанных классов и функций.

Листинг 3.1 — Программная реализация вспомогательных структур

```
class Matrix {
  public:
  int rows;
  int columns;
  int un_zero = 0;
  int** array;

  Matrix() : rows(0), columns(0), array(nullptr) {}

  Matrix(int size) {
    this->rows = size;
    this->columns = size;
    array = new int* [size];
    for (int i = 0; i < size; i++) {
      array[i] = new int[size];
      for (int j = 0; j < size; j++) {
        array[i][j] = 0;
      }
    }
  }

  Matrix(int rows, int columns) {
    this->rows = rows;
    this->columns = columns;
    array = new int* [rows];
    for (int i = 0; i < rows; i++) {
```

```
      array[i] = new int[columns];
      for (int j = 0; j < columns; j++) {
        array[i][j] = 0;
      }
    }
  }

  ~Matrix() {
    clear();
  }

  void print() {
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        cout << setw(4) << array[i][j] << " ";
      }
      cout << "\n";
    }
    cout << "ROWS: " << rows << "\n";
    cout << "COLUMNS: " << columns << "\n";
    cout << "UN_ZERO: " << un_zero << "\n";
  }

  Matrix& operator= (const Matrix& rhs) {
    if (this == (&rhs)) {
      cout << "slefassignment" << "\n";
      return *this;
    }
    clear();

    this->un_zero = rhs.un_zero;
    this->rows = rhs.rows;
    this->columns = rhs.columns;
    array = new int* [rows];

    for (int i = 0; i < rows; i++) {
      array[i] = new int[columns];
      for (int j = 0; j < columns; j++) {
        array[i][j] = rhs.array[i][j];
      }
    }
```

```
68      return *this;
69    }
70
71    void clear() {
72      if (array != nullptr) {
73        for (int i = 0; i < rows; i++) {
74          delete[] array[i];
75        }
76        delete[] array;
77        array = nullptr;
78      }
79    }
80  };
81
82  struct CompressedMatrix {
83    int* AN = nullptr;
84    int* D = nullptr;
85    int len_AN = 0;
86    int size = 0;
87
88    CompressedMatrix() {}
89
90    CompressedMatrix(int* AN, int* D, int len_AN, int size) {
91      this->AN = new int[len_AN];
92      for (int i = 0; i < len_AN; i++) {
93        this->AN[i] = AN[i];
94      }
95      this->D = new int[size];
96      for (int i = 0; i < size; i++) {
97        this->D[i] = D[i];
98      }
99      this->len_AN = len_AN;
100     this->size = size;
101   }
102
103   ~CompressedMatrix() {
104     clear();
105   }
106
107   void print() {
108     cout << "AN: ";
```

```
109     for (int i = 0; i < len_AN; i++) {
110       cout << AN[i] << " ";
111     }
112     cout << endl << "D: ";
113     for (int i = 0; i < size; i++) {
114       cout << D[i] << " ";
115     }
116   }
117
118   CompressedMatrix& operator= (const CompressedMatrix& rhs) {
119     if (this == (&rhs)) {
120       cout << "slefassignment" << "\n";
121       return *this;
122     }
123     clear();
124
125     this->size = rhs.size;
126     this->len_AN = rhs.len_AN;
127     this->AN = new int[len_AN];
128     this->D = new int[size];
129
130     for (int i = 0; i < len_AN; i++) {
131       AN[i] = rhs.AN[i];
132     }
133     for (int i = 0; i < size; i++) {
134       D[i] = rhs.D[i];
135     }
136     return *this;
137   }
138
139   void clear() {
140     if (AN != nullptr) {
141       delete[] AN;
142       AN = nullptr;
143     }
144     if (D != nullptr) {
145       delete[] D;
146       D = nullptr;
147     }
148   }
149 };
```

```
150
151 class KRMCompressedMatrix {
152   public:
153   int* AN = nullptr;
154   int* NR = nullptr;
155   int* NC = nullptr;
156   int* JR = nullptr;
157   int* JC = nullptr;
158   int len_AN = 0;
159   int rows = 0;
160   int columns = 0;
161
162   KRMCompressedMatrix() {}
163
164   KRMCompressedMatrix(Matrix* mat) {
165     this->len_AN = mat->un_zero;
166     this->rows = mat->rows;
167     this->columns = mat->columns;
168     AN = new int[len_AN];
169     NR = new int[len_AN];
170     NC = new int[len_AN];
171     JR = new int[rows];
172     JC = new int[columns];
173     for (int i = 0; i < len_AN; i++) {
174       AN[i] = 0;
175       NR[i] = -1;
176       NC[i] = -1;
177     }
178     for (int i = 0; i < rows; i++) {
179       JR[i] = -1;
180     }
181     for (int i = 0; i < columns; i++) {
182       JC[i] = -1;
183     }
184     create_arrays(mat, AN, NR, NC, JR, JC);
185   }
186
187   KRMCompressedMatrix(map<pair<int, int>, int>* coords, int rows, int
        columns) {
188     this->len_AN = (*coords).size();
189     this->rows = rows;
```

```
190       this->columns = columns;
191
192       AN = new int[len_AN];
193       NR = new int[len_AN];
194       NC = new int[len_AN];
195       JR = new int[rows];
196       JC = new int[columns];
197
198       for (int i = 0; i < len_AN; i++) {
199         AN[i] = 0;
200       }
201       for (int i = 0; i < rows; i++) {
202         JR[i] = -1;
203       }
204       for (int i = 0; i < columns; i++) {
205         JC[i] = -1;
206       }
207
208       create_arrays(coords, AN, NR, NC, JR, JC, rows, columns);
209     }
210
211     KRMCompressedMatrix(int len_AN, int rows, int columns, int*& AN, int
          *& NR, int*& NC, int*& JR, int*& JC) {
212       this->len_AN = len_AN;
213       this->rows = rows;
214       this->columns = columns;
215       this->AN = AN;
216       this->NR = NR;
217       this->NC = NC;
218       this->JR = JR;
219       this->JC = JC;
220     }
221
222     ~KRMCompressedMatrix() {
223       clear();
224     }
225
226     void print() {
227       cout << "AN: ";
228       for (int i = 0; i < len_AN; i++) {
229         cout << setw(3) << AN[i] << " ";
```

```
230     }
231
232     cout << endl << "NR: ";
233     for (int i = 0; i < len_AN; i++) {
234       cout << setw(3) << NR[i] << " ";
235     }
236
237     cout << endl << "NC: ";
238     for (int i = 0; i < len_AN; i++) {
239       cout << setw(3) << NC[i] << " ";
240     }
241
242     cout << endl << "JR: ";
243     for (int i = 0; i < rows; i++) {
244       cout << setw(3) << JR[i] << " ";
245     }
246
247     cout << endl << "JC: ";
248     for (int i = 0; i < columns; i++) {
249       cout << setw(3) << JC[i] << " ";
250     }
251   }
252
253   KRMCompressedMatrix& operator= (const KRMCompressedMatrix& rhs) {
254     if (this == (&rhs)) {
255       cout << "selfassignment" << "\n";
256       return *this;
257     }
258     clear();
259
260     this->rows = rhs.rows;
261     this->columns = rhs.columns;
262     this->len_AN = rhs.len_AN;
263     this->AN = new int[len_AN];
264     this->NR = new int[len_AN];
265     this->NC = new int[len_AN];
266     this->JR = new int[rows];
267     this->JC = new int[columns];
268
269     for (int i = 0; i < len_AN; i++) {
270       this->AN[i] = rhs.AN[i];
```

```
271         this->NR[i] = rhs.NR[i];
272         this->NC[i] = rhs.NC[i];
273       }
274       for (int i = 0; i < rows; i++) {
275         this->JR[i] = rhs.JR[i];
276       }
277       for (int i = 0; i < columns; i++) {
278         this->JC[i] = rhs.JC[i];
279       }
280       return *this;
281     }
282
283     void clear() {
284       if (AN != nullptr) {
285         delete[] AN;
286         AN = nullptr;
287       }
288       if (NR != nullptr) {
289         delete[] NR;
290         NR = nullptr;
291       }
292       if (NC != nullptr) {
293         delete[] NC;
294         NC = nullptr;
295       }
296       if (JR != nullptr) {
297         delete[] JR;
298         JR = nullptr;
299       }
300       if (JC != nullptr) {
301         delete[] JC;
302         JC = nullptr;
303       }
304     }
305 };
```

Листинг 3.2 — Программная реализация вспомогательных функций

```
1 Matrix usual_addition(Matrix* _1, Matrix* _2) {
2   if ((_1->rows != _2->rows) or (_2->columns != _1->columns)) {
3     cout << "Uncampatable sizes in addition: \n";
4     exit(1);
5   }
```

```
  Matrix ans(_1->rows, _1->columns);
  for (int i = 0; i < (*_1).rows; i++) {
    for (int j = 0; j < (*_2).rows; j++) {
      ans.array[i][j] = _1->array[i][j] + _2->array[i][j];
    }
  }
  return ans;
}

Matrix usual_multiplication(Matrix* array_l, Matrix* array_r) {
  Matrix new_ar(array_l->rows, array_r->columns);
  if (array_l->columns != array_r->rows) {
    cout << "Uncampatable sizes in multiplication: \n";
    exit(1);
  }

  for (int i = 0; i < array_l->rows; i++) {

    for (int j = 0; j < array_r->columns; j++) {
      for (int k = 0; k < array_l->columns; k++) {
        new_ar.array[i][j] += (array_l->array[i][k] * array_r->array[k
            ][j]);
      }
      if (new_ar.array[i][j] != 0) new_ar.un_zero += 1;
    }
  }
  return new_ar;
}

int* parse(string str, int len, Matrix* mat) {
  int col = 0;
  for (char i : str)
  if (i == ';') col++;
  string* arr = new string[col];
  int iter = 0;
  for (char i : str) {
    if (i == ';') {
      iter++;
      continue;
    }
    arr[iter] += i;
```

```
46    }
47    if (iter == len) {
48      int* ans = new int[iter];
49      for (int i = 0; i < iter; i++) {
50        ans[i] = str_int(arr[i]);
51        if (ans[i] != 0) mat->un_zero += 1;
52      }
53      delete[] arr;
54      return ans;
55    }
56    else {
57      cout << "There are not enough numbers for a matrix of this size";
58      exit(1);
59    }
60
61    Matrix read_matrix(string filename) {
62      ifstream file(filename);
63      string data;
64      int rows = 0;
65      int r = 0;
66      int c = 0;
67      if (file.is_open())
68      {
69        if (getline(file, data)) {
70          if (data[0] == 'S') {
71            read_len(data, &r, &c);
72          }
73        }
74        Matrix matrix(r, c);
75        // start reading the matrix
76        while (getline(file, data)) {
77          matrix.array[rows] = parse(data, c, &matrix);//////
78          if (rows > r) {
79            cout << "There are too many strings\n";
80            exit(1);
81          }
82          rows++;
83        }
84        if (rows != r) {
85          cout << "Amount of strings is not enoght\n";
86          exit(1);
```

```
87        }
88        return matrix;
89      }
90      else {
91        cout << "File read error\n";
92        exit(1);
93      }
94    }
95
96    void COMPARE(Matrix* _1, Matrix* _2) {
97      for (int i = 0; i < _1->rows; i++) {
98        for (int j = 0; j < _1->columns; j++) {
99          if (_1->array[i][j] != _2->array[i][j]) {
100           cout << "POSITOIN [" << i << "][" << j << "]: " << _1->array[
                i][j] << " != " << _2->array[i][j] << "\n";
101         }
102       }
103     }
104   }
```

Листинг 3.3 — Программная реализация алгоритма упавковки по Дженнингсу

```
1   void create_AN(int** matrix, int size, int* an_size, int** D, int**
        AN) {
2     vector<int> an_loc;
3
4     int glob_ind = 0;
5     int last_ind = 0;
6
7     for (int i = 0; i < size; i++) {
8       (*D)[i] = glob_ind;
9       last_ind = i;
10      for (int j = i; j < size; j++) {
11        if (matrix[i][j] != 0) last_ind = j;
12      }
13      for (int j = i; j < (last_ind + 1); j++) {
14        an_loc.push_back(matrix[i][j]);
15      }
16      glob_ind += (last_ind - i) + 1;
17    }
18
19    (*an_size) = an_loc.size();
20    (*AN) = new int[(*an_size)];
```

```
    for (int i = 0; i < (*an_size); i++) {
      (*AN)[i] = an_loc[i];
    }
  }

  CompressedMatrix matrix_packagin(Matrix* mat) {
    int AN_size = 0;
    int* D = new int[(*mat).rows];
    int* AN = nullptr;

    create_AN((*mat).array, (*mat).rows, &AN_size, &D, &AN);

    CompressedMatrix A(AN, D, AN_size, (*mat).rows);
    delete[] AN;
    delete[] D;
    return A;
  }
```

Листинг 3.4 — Программная реализация алгоритма распаковки из схемы Дженнингса

```
Matrix matrix_unpackagin(CompressedMatrix* comp_mat) {
  Matrix ans(comp_mat->size);
  ans.un_zero = comp_mat->len_AN;
  cout << endl;

  int row = -1;
  int column = -1;
  int ind_D = 0;
  for (int i = 0; i < comp_mat->len_AN; i++) {
    column++;
    if (i == comp_mat->D[ind_D]) {
      row++;
      column = row;
      ind_D++;
    }
    ans.array[row][column] = comp_mat->AN[i];
    ans.array[column][row] = comp_mat->AN[i];
  }
  return ans;
}
```

Листинг 3.5 — Программная реализация алгоритма сложения мартиц сажтаых по схеме Дженнингса

```
CompressedMatrix sum_pack_matrix(CompressedMatrix* _1, CompressedMatrix* _2) {
  vector<int> *loc_sum = new vector<int>[_1->size];
  vector<int> result_AN;
  vector<int> result_D;
  int len_row_1 = 0;
  int len_row_2 = 0;
  int iter_row = 0;

  while (iter_row < (_1->size - 1)) {//size==rows
    len_row_1 = _1->D[iter_row + 1] - _1->D[iter_row];
    len_row_2 = _2->D[iter_row + 1] - _2->D[iter_row];
    for (int i = 0; i < max(len_row_1, len_row_2); i++) {
      loc_sum[iter_row].push_back(0);
      if (i < len_row_1) loc_sum[iter_row][loc_sum[iter_row].size() - 1] += _1->AN[_1->D[iter_row] + i];
      if (i < len_row_2) loc_sum[iter_row][loc_sum[iter_row].size() - 1] += _2->AN[_2->D[iter_row] + i];
    }
    iter_row++;
  }
  loc_sum[iter_row].push_back(0);
  loc_sum[iter_row][loc_sum[iter_row].size() - 1] += _1->AN[_1->D[iter_row]] + _2->AN[_2->D[iter_row]];
  int last_ind = 1;

  for (int i = 0; i < _1->size; i++) {
    result_D.push_back(result_AN.size());
    for (int el = 0; el < loc_sum[i].size(); el++) {
      if (loc_sum[i][el] != 0) last_ind = el;
    }
    for (int j = 0; j <= last_ind; j++) {
      result_AN.push_back(loc_sum[i][j]);
    }
  }

  int* D = new int[_1->size];
  int* AN = new int[result_AN.size()];
  for (int i = 0; i < result_AN.size(); i++) {
    AN[i] = result_AN[i];
```

```
  }
  for (int i = 0; i < _1->size; i++) {
    D[i] = result_D[i];
  }

  CompressedMatrix ans(AN, D, result_AN.size(), _1->size);
  return ans;
}
```

Листинг 3.6 — Программная реализация алгоритма упаковки матрицы по схеме Рейнбольдта-Местеньи

```
int* get_IA(KRMCompressedMatrix* _1) {//rows numbers
  int* IA = new int[_1->len_AN];
  for (int i = 0; i < _1->len_AN; i++) {
    IA[i] = -1;
  }
  int cur;
  for (int i = 0; i < _1->len_AN; i++) {
    if (IA[i] == -1) {
      for (int r = 0; r < _1->rows; r++) {
        if (i == _1->JR[r]) {
          IA[i] = r;
          cur = _1->NR[i];
          while (cur != _1->JR[r]) {
            IA[cur] = r;
            cur = _1->NR[cur];
            //if (cur >= _1->len_AN) cout << "294!!!!!!!!!!!!!!!!!\n"
                ;
          }
        }
      }
    }
  }
  return IA;
}

int*& get_JA(KRMCompressedMatrix* _1) {// get columns numbers
  int* JA = new int[_1->len_AN];
  for (int i = 0; i < _1->len_AN; i++) {
    JA[i] = -1;
  }
  int cur;
```

```
31      for (int i = 0; i < _1->len_AN; i++) {
32        if (JA[i] == -1) {
33          for (int c = 0; c < _1->columns; c++) {
34            if (i == _1->JC[c]) {
35              JA[i] = c;
36              cur = _1->NC[i];
37              while (cur != _1->JC[c]) {
38                JA[cur] = c;
39                cur = _1->NC[cur];
40                //if (cur >= _1->len_AN) cout
41              }
42            }
43          }
44        }
45      }
46      return JA;
47    }
48
49    void create_arrays(Matrix* mat, int*& AN, int*& NR, int*& NC, int*&
          JR, int*& JC)
50    {
51      int R = mat->rows;
52      int C = mat->columns;
53      int LEN_AN = mat->un_zero;
54      int iter = -1;
55      int cur_R = 0;
56      int cur_C = 0;
57      int loc_iter = 0;
58      for (int i = 0; i < R; i++) {
59        for (int j = 0; j < C; j++) {
60          if (mat->array[i][j] != 0) {
61            iter++;
62            if (iter >= LEN_AN) {
63              exit(1);
64            }
65            AN[iter] = mat->array[i][j];
66            NR[iter] = iter;
67            NC[iter] = iter;
68
69            if ((JR[i] == -1) or (JC[j] == -1)) {
70              if (JR[i] == -1) JR[i] = iter;
```

```
            if (JC[j] == -1) JC[j] = iter;
          }
          //NR
          cur_R = JR[i];
          while (cur_R < iter) {
            NR[cur_R] = cur_R + 1;
            cur_R += 1;
          }
          NR[iter] = JR[i];

          //NC
          cur_C = JC[j];
          while (cur_C < iter) {
            if (NC[cur_C] == JC[j]) {
              NC[cur_C] = iter;
              cur_C = iter;
            }
            else if (NC[cur_C] != iter) {
              cur_C = NC[cur_C];
            }
            else {
              NC[cur_C] = iter;
              cur_C = NC[cur_C];//iter
            }
          }
          NC[iter] = JC[j];
        }
      }
    }
  }
```

Листинг 3.7 — Программная реализация алгоритма распкаовки мартиц сажтаых по схеме Рейнбольдта-Местеньи

```
  Matrix unpacking_KRM(KRMCompressedMatrix* krm) {
    int* IA = get_IA(krm);
    int* JA = get_JA(krm);

    Matrix res(krm->rows, krm->columns);
    for (int i = 0; i < krm->len_AN; i++) {
      res.array[IA[i]][JA[i]] = krm->AN[i];
    }
    res.un_zero = krm->len_AN;
```

```
    if (IA != nullptr) delete[] IA;
    if (JA != nullptr) delete[] JA;
    return res;
  }
```

Листинг 3.8 — Программная реализация алгоритма сложения мартиц сажтаых по схеме Рейнбольдта-Местеньи

```
  KRMCompressedMatrix KRM_addition(KRMCompressedMatrix* _1,
     KRMCompressedMatrix* _2) {
    if (_1->columns != _2->columns or _1->rows != _2->rows) {
      cout << "Uncampatable sizes in addition: \n";
      exit(1);
    }
    int rows = max(_1->rows, _2->rows);
    int columns = max(_1->columns, _2->columns);

    // coords arrays
    int* IA_1 = get_IA(_1);
    int* JA_1 = get_JA(_1);


    int* IA_2 = get_IA(_2);
    int* JA_2 = get_JA(_2);

    map <pair<int, int>, int> coords_3;
    for (int i = 0; i < _1->len_AN; i++) {
      if (coords_3.find({IA_1[i], JA_1[i]}) != coords_3.end()) {
        coords_3[{IA_1[i], JA_1[i]}] += _1->AN[i];
      }
      else {
        coords_3[{IA_1[i], JA_1[i]}] = _1->AN[i];
      }
    }

    for (int i = 0; i < _2->len_AN; i++) {
      if (coords_3.find({ IA_2[i], JA_2[i] }) != coords_3.end()) {
        coords_3[{IA_2[i], JA_2[i]}] += _2->AN[i];
      }
      else {
        coords_3[{IA_2[i], JA_2[i]}] = _2->AN[i];
      }
    }
```

```
35
36      //removal the zeros
37      for (auto it = coords_3.begin(); it != coords_3.end(); ) {
38        if (it->second == 0) {
39          it = coords_3.erase(it); // removal the element and renewing
                the iterator
40        }
41        else {
42          ++it; // move to the next
43        }
44      }
45
46      int len_AN = coords_3.size();
47
48      cout << "\n";
49
50      KRMCompressedMatrix res(&coords_3, rows, columns);
51      if (JA_1 != nullptr) delete[] JA_1;
52      if (IA_1 != nullptr) delete[] IA_1;
53      if (JA_2 != nullptr) delete[] JA_2;
54      if (IA_2 != nullptr) delete[] IA_2;
55
56      return res;
57    }
```

Листинг 3.9 — реализаия алгоритма умножения матриц сжатых по схеме Рейнбольдта-Местеньи

```
1   KRMCompressedMatrix KRM_multiplication(KRMCompressedMatrix* _1,
        KRMCompressedMatrix* _2) {
2     int* IA_1 = get_IA(_1);
3     int* JA_1 = get_JA(_1);
4
5     // transponiruem
6     int* IA_2 = get_IA(_2);
7     int* JA_2 = get_JA(_2);
8
9     map <pair<int, int>, int> coords_1;
10    for (int i = 0; i < _1->len_AN; i++) {
11      coords_1[{IA_1[i], JA_1[i]}] = _1->AN[i];
12    }
13
14    map <pair<int, int>, int> coords_2;
15    for (int i = 0; i < _2->len_AN; i++) {
```

```
16        coords_2[{IA_2[i], JA_2[i]}] = _2->AN[i];
17      }
18
19      int R = _1->rows;
20      int C = _2->columns;
21
22      int sum = 0;
23      map <pair<int, int>, int> res_coords;
24      for (int i = 0; i < R; i++) {
25        for (int j = 0; j < C; j++) {
26          for (int k = 0; k < _1->columns; k++) {
27            if ((coords_1.find({i, k}) != coords_1.end()) and (coords_2.
                  find({k, j}) != coords_2.end())) {
28              sum += coords_1[{i, k}] * coords_2[{k, j}];
29            }
30          }
31          // find sum
32          if (sum != 0) {
33            res_coords[{i, j}] = sum;
34            sum = 0;
35          }
36        }
37      }
38
39      KRMCompressedMatrix a(&res_coords, R, C);
40
41      if (JA_1 != nullptr) delete[] JA_1;
42      if (IA_1 != nullptr) delete[] IA_1;
43      if (JA_2 != nullptr) delete[] JA_2;
44      if (IA_2 != nullptr) delete[] IA_2;
45      return a;
46    }
```

3.3 Тестирование программы

В таблице 3.1 и 3.2 представлены описания тестов по методологии чёрного ящика, все тесты пройдены успешно.

Таблица 3.1 — Описание тестов по методологии чёрного ящика

	Описание теста	**Входные данные**	**Ожидаемый результат**	**Полученный результат**
1	проверка на обработку не валидных данных	3	оповещение о некорректности данных и запрос новых данных	оповещение о некорректности данных и запрос новых данных
2	проверка на обработку не валидных данных	1	включение режима работы с матрицами через схему Дженнингса	включение режима работы с матрицами через схему Дженнингса
3	проверка на обработку не валидных данных	2	включение режима работы с матрицами через схему Рейнбольдта-Местеньи	включение режима работы с матрицами через схему Рейнбольдта-Местеньи
4	проверка на обработку не валидных данных в Дженнингсе	SIZE;(2, 2); 1;2; 2;2;1;	оповещение о некорректности данных и завершение программы	оповещение о некорректности данных и завершение программы
5	проверка суммирования матриц в виде Дженнингса	Файл 1: 1 0 0 0 0 2 3 0 0 3 4 0 0 0 0 5 Файл 2: 2 3 0 0 3 0 0 4 0 0 1 0 0 4 0 5	первая матрица упакованная: AN: 1 2 3 4 5 D: 0 1 3 4 вторая матрица упакованная: AN: 2 3 0 0 4 1 5 D: 0 2 5 6 результирующая матрица упакованная: AN: 3 3 2 3 4 5 10 D: 0 2 5 6 результирующая матрица: 3 3 0 0 3 2 3 4 0 3 5 0 0 4 0 10	первая матрица упакованная: AN: 1 2 3 4 5 D: 0 1 3 4 вторая матрица упакованная: AN: 2 3 0 0 4 1 5 D: 0 2 5 6 результирующая матрица упакованная: AN: 3 3 2 3 4 5 10 D: 0 2 5 6 результирующая матрица: 3 3 0 0 3 2 3 4 0 3 5 0 0 4 0 10

Таблица 3.2 — Описание тестов по методологии чёрного ящика

	Описание теста	**Входные данные**	**Ожидаемый результат**	**Полученный результат**
6	проверка для схемы Дженнингса на сложение матриц с итоговой нулевой	Файл 1: 0 10 0 0 0 0 0 0 0 Файл 2: 0 -10 0 0 0 0 0 0 0	первая матрица упакованная: AN: 0 10 0 0 D: 0 2 3 вторая матрица упакованная: AN: 0 -10 0 0 D: 0 2 3 результирующая матрица упакованная: AN: 0 0 0 D: 0 1 2 результирующая матрица: 0 0 0 0 0 0 0 0 0	первая матрица упакованная: AN: 0 10 0 0 D: 0 2 3 вторая матрица упакованная: AN: 0 -10 0 0 D: 0 2 3 результирующая матрица упакованная: AN: 0 0 0 D: 0 1 2 результирующая матрица: 0 0 0 0 0 0 0 0 0
7	проверка для КРМ на умножение матриц	Файл 1: 0 10 0 0 0 0 20 80 0 Файл 2: 0 -10 0 0 0 0 -20 -80 0	0 0 0 0 0 0 0 -200 0 упакованная результирующая матрица: AN: -200 NR: 0 NC: 0 JR: -1 -1 0 JC: -1 0 -1	0 0 0 0 0 0 0 -200 0 упакованная результирующая матрица: AN: -200 NR: 0 NC: 0 JR: -1 -1 0 JC: -1 0 -1

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были изучены алгоритмы сжатия матриц, написаны и протестированы их реализации.

Для достижения поставленной цели были успешно выполнены основные задачи:

1) разобраны суть основных понятий используемых для преобразовании и вычислении,
2) разработаны алгоритмы упаковки и распаковки матриц по схеме Дженинггса и Рейнбольдта-Местеньи,
3) разработаны алгоритмы сложения и умножения упакованных матриц,
4) реализованы алгоритмы,
5) проведено тестирование, проверена работоспособность реализаций алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Писсанецки С. Технология разреженных матриц: Пер. с английского – Х. Д. Икрамова. — Москва: Мир, 1988. — С. 22-23.