



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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе №3

Название «Обработка разреженных матриц»

Дисциплина «Типы и структуры данных»

Вариант 5

Студент ИУ7-31Б

(подпись, дата)

Корниенко К. Ю.
(Фамилия И.О.)

Преподаватель

(подпись, дата)

Силантьева А. В.
(Фамилия И.О.)

Москва, 2021

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Способы хранения разреженных матриц	4
1.2 Описание условия задачи	4
1.3 Техническое задание	5
1.4 Вывод	5
2 Конструкторский раздел	6
2.1 Описание структур данных	6
2.2 Описание алгоритма	8
2.3 Вывод	8
3 Технологический раздел	9
3.1 Требование к ПО	9
3.2 Реализация алгоритмов	9
3.3 Тестовые данные	13
3.4 Вывод	13
4 Исследовательский раздел	14
4.1 Постановка эксперимента	14
4.2 Технические характеристики	14
4.3 Анализ временной сложности и затрат памяти	14
5 Контрольные вопросы	18
Заключение	19

Введение

Матрицы широко используются для представления информации о многих сферах деятельности. Матрицы и эффективные алгоритмы работы с ними применяются в анализе данных и машинном обучении. Часто применяют алгоритмы для работы с **разреженными** матрицами, иными словами матрицами, количество нулевых элементов в которой во много раз превышает количество ненулевых.

Целью данной работы является реализация алгоритмов обработки разреженных матриц, сравнение эффективности использования этих алгоритмов (по времени выполнения и по требуемой памяти) со стандартными алгоритмами обработки матриц при различном процентном заполнении матриц ненулевыми значениями и при различных размерах матриц.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать подходы к хранению разреженных матриц;
- описать используемые структуры данных;
- описать алгоритм умножения разреженного вектора на разреженную матрицу;
- протестировать разработанное ПО;
- сравнить эффективность по времени и памяти умножения вектора на матрицу в разреженном и классическом представлениях.

1 Аналитический раздел

В данном разделе будут представлены основные сведения о способах хранения разреженных матриц. Также будет описано условие задачи и техническое задание.

1.1 Способы хранения разреженных матриц

Существуют различные методы хранения элементов матрицы в памяти. Например, линейный связный список, т.е. последовательность ячеек, связанных в определенном порядке. Каждая ячейка списка содержит элемент списка и указатель на положение следующей ячейки. Можно хранить матрицу, используя кольцевой связный список, двунаправленные стеки и очереди. Существует диагональная схема хранения симметричных матриц, а также - связные схемы разреженного хранения. Связная схема хранения матриц, предложенная Кнутом, предлагает хранить в массиве (например, в AN) в произвольном порядке сами элементы, индексы строк и столбцов соответствующих элементов (например, в массивах I и J), номер (из массива AN) следующего ненулевого элемента, расположенного в матрице по строке (NR) и по столбцу (NC), а также номера элементов, с которых начинается строка (указатели для входа в строку - JR) и номера элементов, с которых начинается столбец (указатели для входа в столбец - JC). Данная схема хранения избыточна, но позволяет легко осуществлять любые операции с элементами матрицы.

Наиболее широко используемая схема хранения разреженных матриц - это схема, предложенная Чангом и Густавсоном, называемая: "разреженный строчный формат". Эта схема предъявляет минимальные требования к памяти и очень удобна при выполнении операций сложения, умножения матриц, перестановок строк и столбцов, транспонирования, решения систем линейных уравнений, при хранении коэффициентов в разреженных матрицах и т.п. В этом случае значения ненулевых элементов хранятся в массиве AN, соответствующие им столбцовые индексы - в массиве JA. Кроме того, используется массив указателей, например, IA, отмечающих позиции AN и JA, с которых начинаются описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN.

В связи с тем, что разреженный строчный формат предъявляет минимальные требования к памяти и удобна для операции умножения, в данной лабораторной работе будет использована именно она.

1.2 Описание условия задачи

Разработать программу умножения или сложения разреженных матриц. Предусмотреть возможность ввода данных, как с клавиатуры, так и использования заранее подготовленных данных. Матрицы хранятся и выводятся в форме трех объектов. Для небольших матриц можно дополнительно вывести матрицу в виде матрицы. Величина матриц - любая (допустим, 1000×1000). Сравнить эффективность (по памяти и по времени выполнения) стандартных алгоритмов обработки матриц с алгоритмами обработки разреженных матриц при различной степени разреженности матриц и различной размерности матриц.

1.3 Техническое задание

Разреженная (содержащая много нулей) матрица хранится в форме 3-х объектов:

- вектор A содержит значения ненулевых элементов;
- вектор JA содержит номера столбцов для элементов вектора A ;
- связный список IA , в элементе N_k которого находится номер компонент в A и JA , с которых

начинается описание строки N_k матрицы A .

1) Смоделировать операцию умножения вектора-строки и матрицы, хранящихся в этой форме, с получением результата в той же форме.

2) Произвести операцию умножения, применяя стандартный алгоритм работы с матрицами.

3) Сравнить время выполнения операций и объем памяти при использовании этих 2-х алгоритмов при различном проценте заполнения матриц.

1.4 Вывод

В данном разделе были описаны способы хранения матриц в памяти и сформулировано техническое задание. В результате анализа способов хранения матриц для реализации операции умножения вектора-строки на матрицу был выбран разреженный строчный формат хранения.

2 Конструкторский раздел

В данном разделе будут описаны используемые структуры данных, приведен список функций для работы с данными типами. Также будет описан алгоритм умножения разреженных матриц.

2.1 Описание структур данных

Ниже, на листинге 2.1 представлены коды возможных ошибок.

Листинг 2.1 — Коды ошибок

```
1 #define SUCCESS      0    // Успешное выполнение
2 #define MEM_ERR      1    // Ошибка памяти
3 #define INP_ERR      2    // Ошибка ввода
4 #define BAD_MATRIX   3    // Некорректная матрица
5 #define BAD_PERCENT  4    // Некорректный процент заполнения
6 #define ARGS_ERR     5    // Некорректные аргументы, переданные в функцию
7 #define MUL_ERR      6    // Матрицы невозможно перемножить
8 #define BAD_VECTOR   7    // Вектор некорректен
9 #define BAD_FILE     8    // Файл некорректен
```

Ниже, на листинге 2.2 представлены сокращения используемых типов данных.

Листинг 2.2 — Используемые типы

```
1 typedef double data_t;
2 typedef unsigned int id_t;
```

Ниже, на листинге 2.3 представлена структура связанного списка.

Листинг 2.3 — Связный список и функции для работы с ним

```
1 typedef struct node
2 {
3     id_t col_index;    // индекс столбца
4     struct node* next; // следующий элемент
5 } node_t;
6
7 typedef struct list
8 {
9     node_t* head;     // указатель на голову
10 } list_t;
```

На листинге 2.4, представленном ниже, описана структура разреженной матрицы и функции для работы с ней.

Листинг 2.4 — Разреженная матрица и функции для работы с ней

```
1 typedef struct smatrix
2 {
3     id_t rows;
4     id_t cols;
```

```

5      id_t size;
6
7      data_t* A;
8      id_t* JA;
9      list_t IA;
10 } smatrix_t;
11
12 // Представление разреженной матрицы по-умолчанию
13 smatrix_t smat_null(void);
14
15 // Проверка на корректность разреженной матрицы
16 bool smat_is_valid(const smatrix_t* mat);
17
18 // Освобождение памяти
19 void smat_destroy(smatrix_t* mat);
20
21 // Получение элемента по индексам
22 data_t smat_get(const smatrix_t* mat, id_t row, id_t col);

```

Ниже, на листинге 2.5 представлено описание структуры классической матрицы и функции для работы с ней.

Листинг 2.5 — Классическое представление матрицы и функции для работы с ней

```

1  typedef struct stdmat
2  {
3      id_t rows;
4      id_t cols;
5      data_t** data;
6  } stdmat_t;
7
8  // Инициализация по-умолчанию
9  stdmat_t stdm_null(void);
10
11 // Нулевая матрица заданного размера
12 stdmat_t stdm_zero(id_t rows, id_t cols);
13
14 // Освобождение памяти
15 void stdm_destroy(stdmat_t* mat);
16
17 // Проверка на корректность данных в матрице
18 bool stdm_is_valid(const stdmat_t* mat);
19
20 // Проверка на возможность перемножить две матрицы
21 bool stdm_is_multable(const stdmat_t* left, const stdmat_t *right);
22
23 // Случайная матрица
24 int stdm_randomize(stdmat_t* mat, double zero_percent);

```

Ниже, на листинге 2.6 представлены операции над матрицами.

Листинг 2.6 — Функции умножения матриц

```
1 // Умножение вектора-строки на матрицу в обычной форме
2 int std_mul(stdmat_t *res, const stdmat_t *v, const stdmat_t *m);
3
4 // Умножение вектора-строки на матрицу в разреженной форме
5 int sparse_mul(smatrix_t *res, const smatrix_t *v, const smatrix_t *m);
```

2.2 Описание алгоритма

Для умножения вектора-строки на разреженную матрицу исходная матрица сначала транспонируется, затем применяется операция скалярного произведения вектора-строки на каждую строку из разреженной матрицы. Для сокращения операций хранится расширенный массив указателей IP. При этом умножаются только ненулевые элементы.

2.3 Вывод

В данном разделе были представлены используемые структуры данных, а также описан алгоритм умножения разреженного вектора-строки на разреженную матрицу.

3 Технологический раздел

В данном разделе будут представлены листинги кодов реализации операции умножения двух матриц в классическом и разреженном представлении и произведено тестирование ПО.

3.1 Требование к ПО

К программе предъявлены следующие требования:

- На вход подается режим работы ПО. Данные, подаваемые на вход, зависят от выбранного режима (0 - авто, 1 - ручной);

При авто режиме:

- на вход программе подаются количества столбцов и строк, а также проценты разреженности вектора-строки и матрицы;

При ручном режиме:

- на вход программе подаются количества столбцов и строк, а также ненулевые элементы по их позициям (для вектора-строки и для матрицы);
- на выходе – матрица, которая является результатом умножения входных вектора-строки и матрицы.

3.2 Реализация алгоритмов

Ниже, на листинге 3.1 представлены реализации умножение матриц, представленных классически, и разреженных матриц.

Листинг 3.1 — Реализация умножения матриц

```
1 int std_mul(stdmat_t *res, const stdmat_t *v, const stdmat_t *m)
2 {
3     if (res == NULL)
4         return ARGS_ERR;
5
6     if (!stdm_is_valid(m) || !stdm_is_valid(v))
7         return BAD_MATRIX;
8
9     if (!stdm_is_multable(v, m))
10        return MUL_ERR;
11
12    if (v->rows != 1)
13        return BAD_VECTOR;
14
15    *res = stdm_zero(v->rows, m->cols);
16
17    if (!stdm_is_valid(res))
18        return MEM_ERR;
19
```

```

20     for (unsigned int col = 0; col < v->cols; col++)
21         for (unsigned int row = 0; row < m->rows; row++)
22             res->data[0][col] += v->data[0][row] * m->data[row][col];
23
24     return SUCCESS;
25 }
26
27 static stdmat_t _std_transpose(stdmat_t *src)
28 {
29     stdmat_t res = stdm_zero(src->cols, src->rows);
30
31     for (id_t col = 0; col < src->cols; col++)
32         for (id_t row = 0; row < src->rows; row++)
33             res.data[col][row] = src->data[row][col];
34
35     return res;
36 }
37
38 static smatrix_t _transpose(const smatrix_t *src)
39 {
40     stdmat_t res = stdm_null();
41     smatrix_t sres = smat_null();
42
43     smat_to_stdmat(&res, src);
44     stdmat_t rest = _std_transpose(&res);
45     stdm_to_smat(&sres, &rest);
46
47     stdm_destroy(&res);
48     stdm_destroy(&rest);
49
50     return sres;
51 }
52
53 static smatrix_t _preinit_rowvec(id_t size)
54 {
55     smatrix_t res = smat_null();
56
57     res.rows = 1;
58     res.cols = size;
59
60     res.A = malloc(size * sizeof(data_t));
61     if (res.A != NULL)
62     {
63         res.JA = malloc(size * sizeof(id_t));
64         if (res.JA != NULL)
65         {
66             res.IA = lst_reserve(2, 0);

```

```

67         if (res.IA.head != NULL)
68             return res;
69
70         free(res.JA);
71     }
72
73     free(res.A);
74 }
75
76 return smat_null();
77 }
78
79 int sparse_mul(smatrix_t *res, const smatrix_t *v, const smatrix_t *m)
80 {
81     if (res == NULL)
82         return ARGS_ERR;
83
84     if (!smat_is_valid(v) || !smat_is_valid(m))
85         return BAD_MATRIX;
86
87     if (v->rows != 1 || v->cols != m->rows)
88         return MUL_ERR;
89
90     *res = _preinit_rowvec(m->cols);
91     if (!smat_is_valid(res))
92         return MEM_ERR;
93
94     int *IP = malloc(v->cols * sizeof(int));
95     if (IP == NULL)
96     {
97         smat_destroy(res);
98         return MEM_ERR;
99     }
100
101     for (id_t i = 0; i < v->cols; i++)
102         IP[i] = -1;
103
104     for (id_t i = 0; i < v->size; i++)
105         IP[v->JA[i]] = i;
106
107     smatrix_t mt = _transpose(m);
108     node_t *IA_node = mt.IA.head;
109     for (id_t row = 0; row < mt.rows; row++)
110     {
111         // index - позиция первого ненулевого элемента в массиве A в строке row м
            атрицы mt

```

```

112     // index_last - позиция первого ненулевого элемента в массиве A в следующ
    ей строке за row
113     id_t index = IA_node->col_index;
114     id_t index_last = IA_node->next->col_index;
115
116     data_t sum = 0;
117
118     // цикл по всем ненулевым элементам строки row матрицы mt
119     for (; index < index_last; index++)
120     {
121         int col = IP[mt.JA[index]];
122         // если соответствующий элемент в IP не равен -1 -> умножаем
123         if (col != -1)
124             sum += mt.A[index] * v->A[col];
125     }
126
127     // установка результата в результирующий вектор
128     if (sum != 0)
129     {
130         res->A[res->size] = sum;
131         res->JA[res->size] = row;
132         res->size++;
133     }
134
135     IA_node = IA_node->next;
136 }
137
138 // обновление последнего элемента в списке IA
139 res->IA.head->next->col_index = res->size;
140
141 smat_destroy(&mt);
142 free(IP);
143 return SUCCESS;
144 }

```

3.3 Тестовые данные

В таблице 3.1 приведены тесты для функции умножения разреженных вектора-строки на матрицу.

Вектор-строка	Матрица	Ожидаемый результат
$(0 \ a \ b)$	$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 3 \\ 0 & 0 & 0 \end{pmatrix}$	Сообщение об ошибке
$(0 \ 0 \ 9)$	$\begin{pmatrix} 0 & 0 & 9 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{pmatrix}$	$(0 \ 0 \ 54.0)$
$(1 \ 0 \ 9 \ 0 \ 0 \ 0)$	$\begin{pmatrix} 3 & 2 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 7 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}$	$(3 \ 2 \ 0)$
$(7 \ 0 \ 3)$	$\begin{pmatrix} 2 \\ 0 \\ 7 \end{pmatrix}$	(35.0)

Таблица 3.1 — Тестовые данные

3.4 Вывод

В данном разделе были разработаны исходные коды алгоритмов: умножение матриц в классическом представлении и в разреженном виде.

4 Исследовательский раздел

В данном разделе будут проведено сравнение работы алгоритмов умножения матриц и разреженных матриц, а также представлены графики сравнительного анализа.

4.1 Постановка эксперимента

Объектом для постановки эксперимента является влияние размерности матрицы на время работы алгоритма и процента заполненности матрицы на объем памяти, ей занимаемой, при фиксированном значении размерности матрицы ($N = 100$) сравнение временной эффективности умножения матриц в классическом представлении и в разреженном виде.

Эксперимент проводится на квадратных матрицах размером от 50×50 до 300×300 с шагом 50. Планируется сделать 50 замеров, на основании которых результат для каждой размерности будет усреднен.

4.2 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- операционная система: Ubuntu Linux 20.04 64-bit;
- оперативная память: 16 GB;
- процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx @ 8x 2,1GHz.

4.3 Анализ временной сложности и затрат памяти

Ниже, на рисунках 4.1 и 4.2 представлена зависимость времени умножения матриц от размерности массива для классического представления матриц и для разреженных матриц. А на рисунках 4.3 и 4.4 представлена зависимость объема занимаемой памяти для двух типов представления матриц от процента заполненности (для матриц 100×100 и 10×10).

Исходя из приведенных ниже графиков можно сделать вывод, что умножение матриц в разреженном строчном формате эффективнее обычного по времени только если процент разреженности матрицы превышает 30-40%. Если процент заполненности матрицы больше 65-70% при размерности 100×100 , то разреженное представление занимает больше памяти, чем представление матрицы в классическом виде, а при размерности 10×10 разреженная форма представления становится неэффективна по памяти уже при 5-7% заполнения.

Рисунок 4.1 — График зависимости времени от размерности матрицы при разреженности 20%

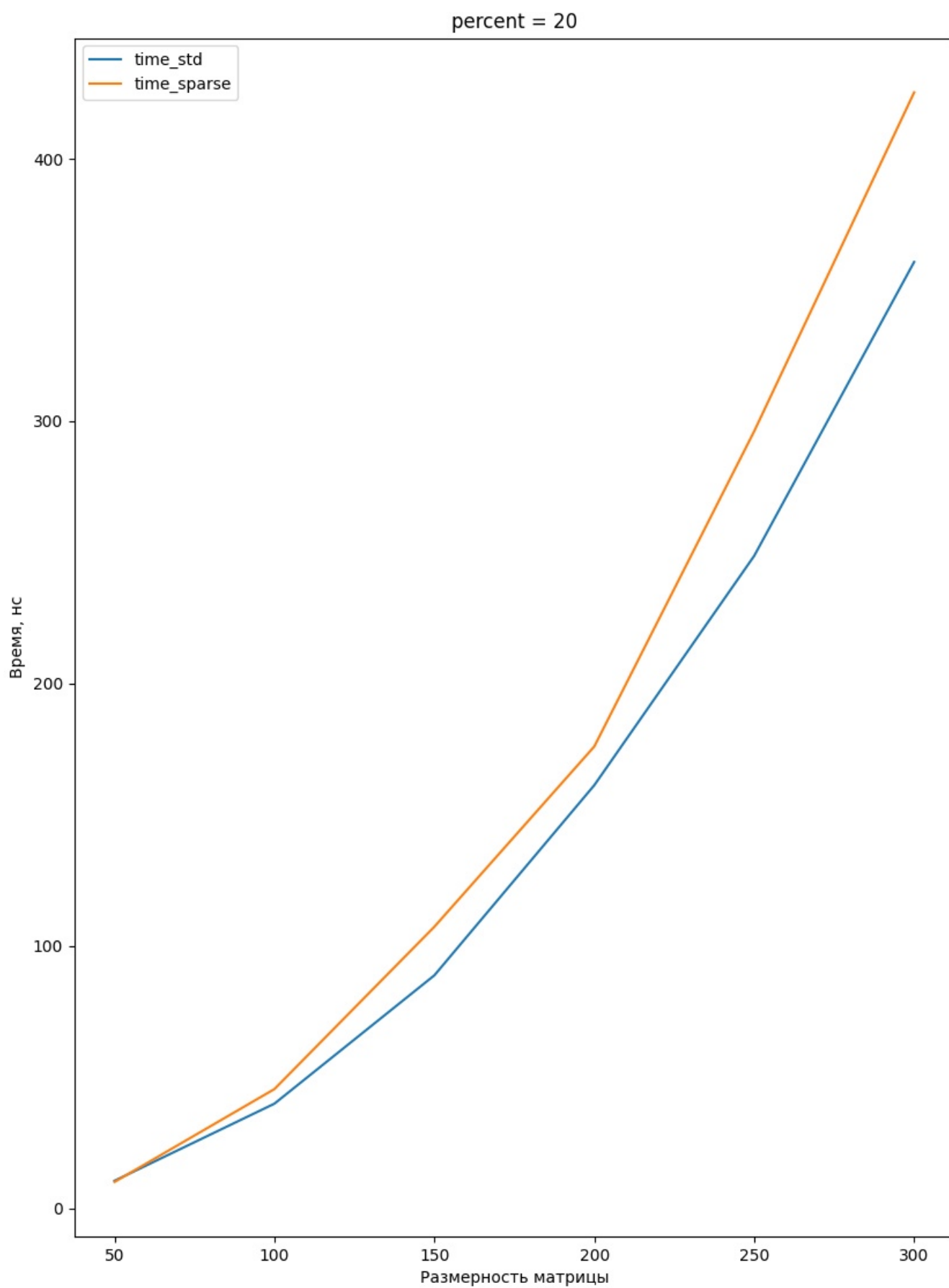


Рисунок 4.2 — График зависимости времени от размерности матрицы при разреженности 80%

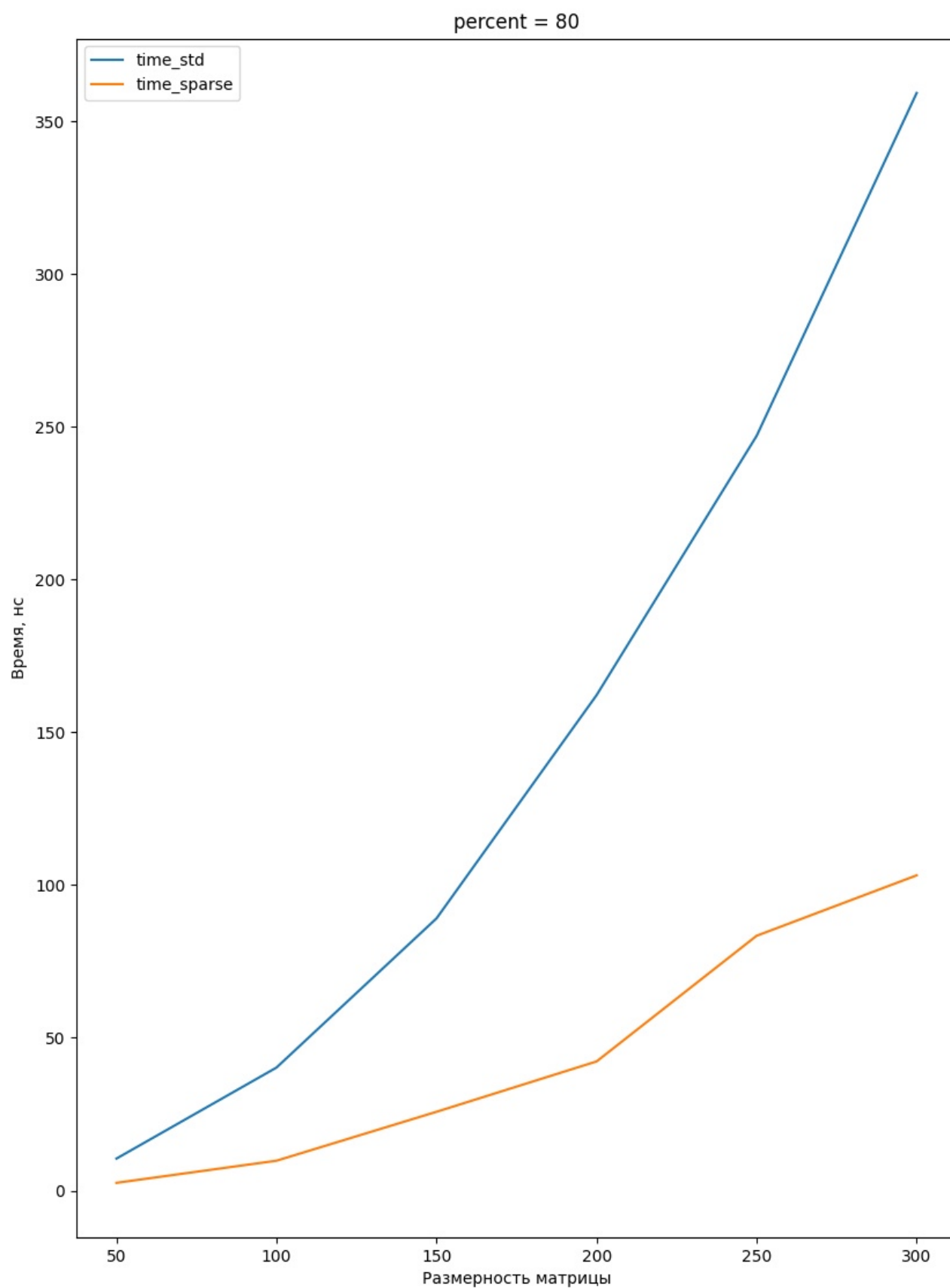


Рисунок 4.3 — График зависимости занимаемой матрицей памяти от процента заполненности (100×100)

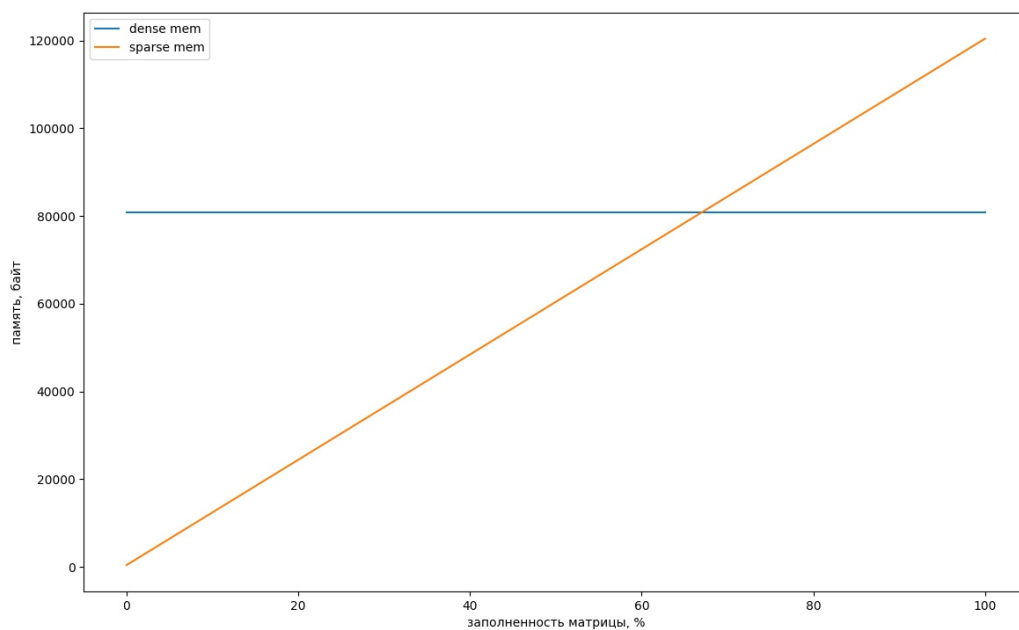
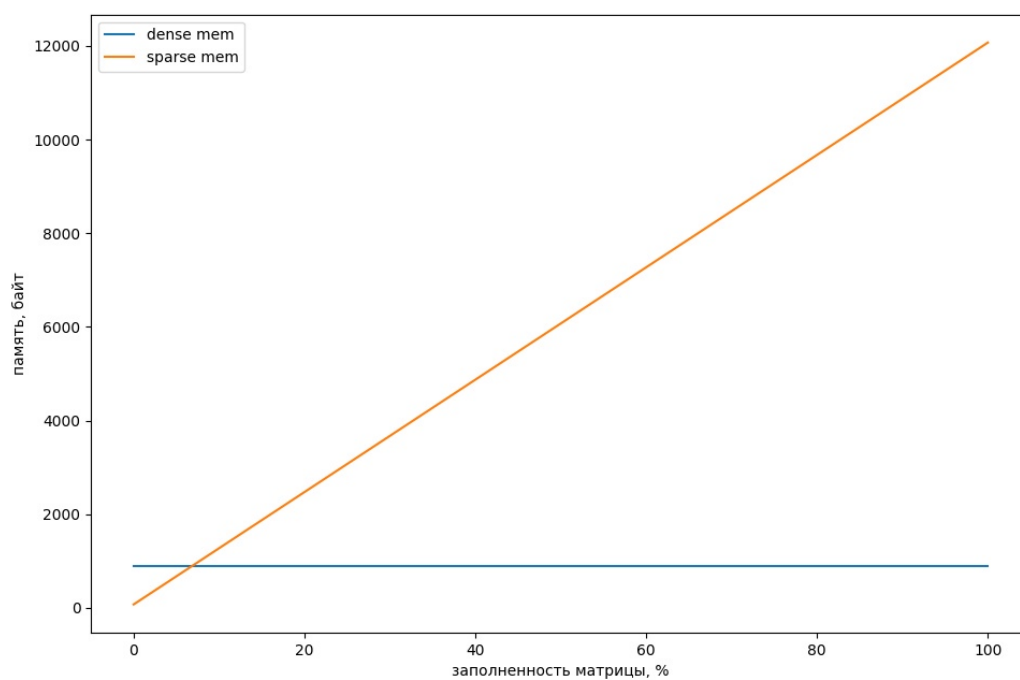


Рисунок 4.4 — График зависимости занимаемой матрицей памяти от процента заполненности (10×10)



5 Контрольные вопросы

1) Что такое разреженная матрица, какие схемы хранения таких матриц Вы знаете?

Разреженная матрица - матрица, количество ненулевых элементов в которой намного меньше, чем количество нулевых.

Разреженная матрица это структура данных, в которой хранятся только ненулевые элементы матрицы, и информация об их позиции в матрице. Такой информацией может быть, например явное указание строки и столбца (координатная форма), а может быть только индекс строки, но вместе с ненулевыми элементами тогда хранится список индексов элементов с которых начинается тот или иной столбец в матрице (Йельский формат). Также, в ряде случаев работа происходит только с симметричными матрицами. Тогда нам достаточно хранить только половину от всех ненулевых элементов матрицы. Существует и множество других форматов, которые разрабатывались для определённой конфигурации матриц, и подходящие для очень узкого круга задач, например, можно хранить матрицу блоками.

2) Каким образом и сколько памяти выделяется под хранение разреженной и обычной матрицы?

Для хранения матрицы в обычном представлении память выделяется сразу под все элементы матрицы. Для хранения разреженной матрицы память выделяется по мере необходимости и только для ненулевых элементов матрицы. При этом, для хранения одного элемента в разреженном формате требуется больше памяти, чем в обычном. Тем не менее, при малой заполненности матрицы хранение только ненулевых элементов становится выгоднее.

3) Каков принцип обработки разреженной матрицы?

Принцип обработки разреженной матрицы заключается в том, чтобы обходить только ненулевые элементы матрицы, а не все возможные, тем самым облегчая сложность алгоритма с $O(N^2)$ до $O(K)$ где N - размерность матрицы, а K - число ненулевых элементов в ней.

4) В каком случае для матриц эффективнее применять стандартные алгоритмы обработки матриц? От чего это зависит?

Это зависит от выбранного формата хранения разреженной матрицы, а также в меньшей степени от процента заполненности матрицы. Чем он меньше (разреженность выше), тем эффективнее использование алгоритмов, работающих с разреженными матрицами. Однако, если процент разреженности матрицы не превосходит 30 – 40% то стандартные алгоритмы обработки оказываются не только проще, но и эффективнее нестандартных.

Заключение

В результате выполнения лабораторной работы были исследованы подходы к хранению разреженных матриц, спроектированы и реализованы структуры данных для их хранения. Был описан алгоритм умножения разреженного вектора на разреженную матрицу и протестированно разработанное ПО.

В ходе тестирования ПО было выявлено, что использование разреженной формы представления матриц выгодно для обработки при большом количестве элементов и проценте разреженности более 30-40%. При размерности матрицы 100×100 разреженный строчный формат становится неэффективным по памяти при 65-70% заполненности, в то время как матрицы 10×10 неэффективны уже при 5-7%.