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»

Отчет по лабораторной работе № 5 по курсу "Анализ алгоритмов"

Тема Организация асинхронного взаимодействия между потоками на примере моделирования конвейера

Студент Кузнецова А. В.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2022 г.

Оглавление

Введение

Существуют задачи, в которых разные алгоритмы обрабатывают один и тот же набор данных друг за другом. При этом, может стоять задача обработки большого объема данных. Для ускорения решения таких вычислительных задач используется конвейерная обработка.

Идея конвейерной обработки [1] заключается в выделении отдельных этапов выполнения общей операции. Каждый этап, выполнив свою работу, передает результат следующему, одновременно принимая новую порцию входных данных. Отдельный этап называют лентой. При таком способе организации вычислений увеличивается скорость обработки за счет совмещения прежде разнесенных во времени операций.

В многопоточном программировании конвейерная обработка реализуется следующим образом: под каждую ленту конвейера выделяется отдельный поток. Выделенные потоки работают асинхронно.

Целью данной лабораторной работы является получение навыка организации асинхронного взаимодействия между потоками на примере моделирования конвейера обработки разреженных матриц, представленных с помощью кольцевой КРМ-схемы. Для достижения поставленной цели требуется выполнить следующие задачи.

1. Описать КРМ-схему хранения матриц.
2. Разработать и реализовать последовательный алгоритм.
3. Выделить алгоритмы, выполняемые на отдельных лентах конвейера.
4. Разработать и реализовать конвейерную версию данного алгоритма.
5. Создать схемы изучаемых алгоритмов.
6. Определить средства программной реализации.
7. Выполнить замеры процессорного времени работы реализаций алгоритма.
8. Провести сравнительный анализ по времени работы реализаций алгоритмов.

9. Подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе представлено теоретическое описание КРМ-схемы хранения разреженных матриц и алгоритмов обработки матриц.

1.1 КРМ-схема хранения разреженных матриц

Для начала следует описать схему хранения разреженной матрицы, предложенную Кнутом. Ненулевые элементы хранятся в компактной форме в одномерном массиве AN. Информация о положении ненулевых элементов в матрице хранится двумя дополнительными параллельными одномерными массивами — I и J; здесь для каждого ненулевого элемента содержатся его строчный и столбцовый индексы. Итак, для каждого $a_{ij} \neq 0$ в памяти находится тройка (a_{ij}, i, j). Далее, чтобы можно было легко отыскивать элементы произвольной строки или столбца матрицы, необходимы еще пара указателей для каждой тройки, а также указатели входа для строк и столбцов, сообщающие начало каждого строчного или столбцового списка. Пусть NR («next nonzero element in the same row» — «следующий ненулевой элемент той же строки») — массив, хранящий строчные указатели, а NC («next nonzero element in the same column» — «следующий ненулевой элемент того же столбца») — массив столбцовых указателей. Пять массивов AN, I, J, NR и NC имеют одинаковую длину, и их одноименные позиции соответствуют друг другу. Пусть JR и JC — массивы, содержащие указатели входа для строк и столбцов, расположенные в соответствии с порядком строк и столбцов матрицы. Тогда рассмотрим матрицу на рис. 1.1, её представление с помощью схемы Кнута — на рис. 1.2.

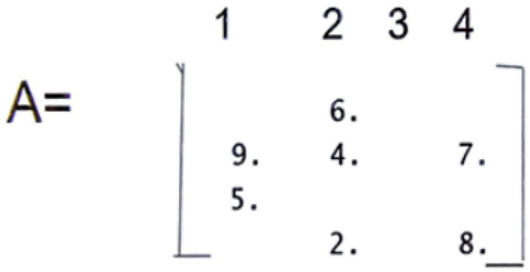

Рисунок 1.1 – Разреженная матрица

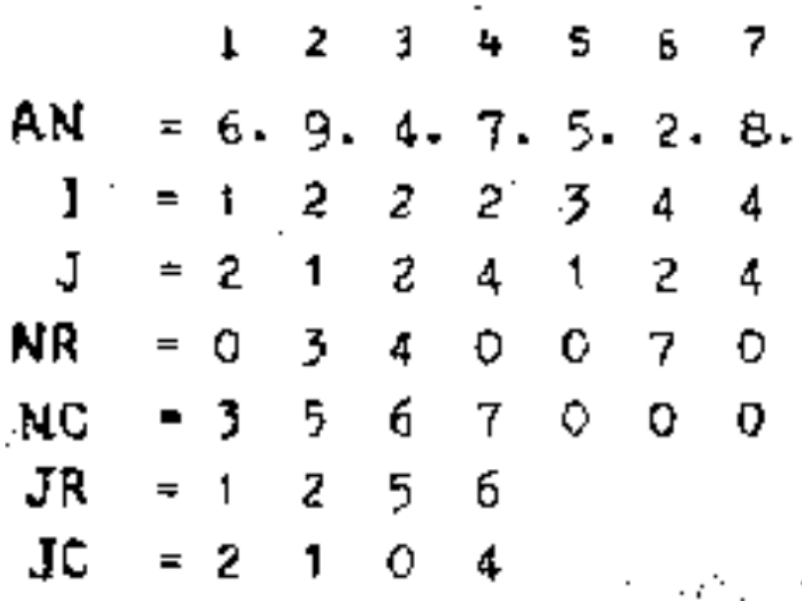

Рисунок 1.2 – Схема Кнута

Рейнболдт и Местеньи предложили модификацию схемы Кнута, сохраняющую ее ценные свойства, но использующую значительно меньше накладных расходов по памяти. Она получила название схемы Кнута-Рейнболдта-Местеньи, или кольцевая КРМ-схема. Связные списки строк и столбцов закольцовываются, а начальные позиции списков включаются в указатели входа. Списки, ассоциированные со строками (столбцами), попарно не пересекаются и потому могут быть совместно хранимы одним массивом NR (для столбцов — NC). Для матрицы на рис. 1.1 [2] приведено ее представление с помощью КРМ-схемы на рис. 1.3. Эта схема более плотная по сравнению со схемой Кнута. Однако, если приходится просматривать элементы некоторой строки (или столбца), то в сжатом формате нет никакой информации о столбцовых (строчных) индексах этих элементов [3].

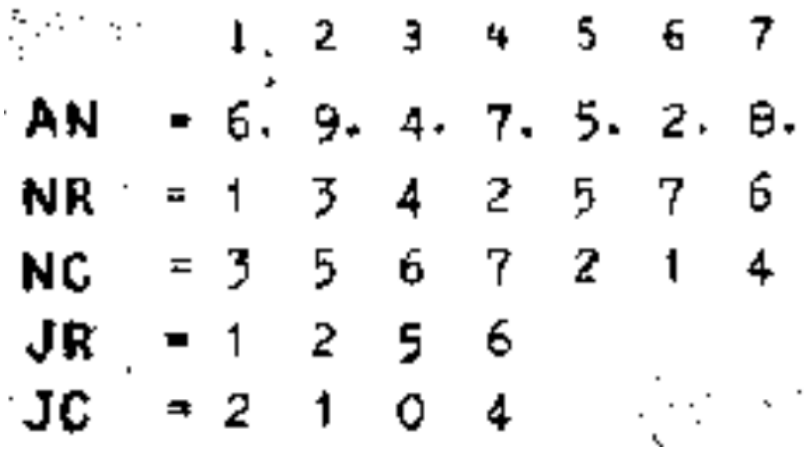

Рисунок 1.3 – Кольцевая КРМ-схема

1.2 Последовательный алгоритм

В некоторых задачах происходит обработка матриц перед их использованием в решении. Данные проходят ряд преобразований в несколько последовательных этапов. Каждый этап реализуется при помощи алгоритма работы с матрицами.

В данной лабораторной работе будет реализована обработка разреженных матриц, состоящая из следующих этапов:

— генерация упакованной верхнетреугольной матрицы в КРМ-схему хранения;

— удаление элементов из матрицы, меньших, либо равных q (q - задано);

— вывод исходной и измененной матрицы в файл.

1.3 Алгоритмы, выполняемые на отдельных лентах

Каждый из описанных выше этапов будет выполняться на отдельной ленте.

1.3.1 Генерация верхнетреугольной упакованной в КРМ-схему хранения матрицы

Генерация верхнетреугольной упакованной в КРМ-схему хранения матрицы будет происходить следующим образом:

1. Значение каждого элемента матрицы, стоящего над главной диагональю и на ней, рандомно генерируется и записывается в массив AN.

2. Если это не последний элемент в строке, то в массив NR записывается (номер текущего элемента + 1).

3. Если сгенерированный элемент — последний элемент в строке, то в массив NR записывается (номер текущего элемента + номер текущей строки — номер текущего столбца).

4. Если элемент находится не на главной диагонали, то в массив NC записывается (номер текущего элемента + размерность матрицы − номер текущей строки).

5. Если элемент находится на главной диагонали, то в массив NC записывается (номер текущего элемента − размерность матрицы · номер текущей строки + (1 + номер текущей строки) · номер текущей строки / 2) и в массив JR записывается номер текущего элемента.

6. Если элемент находится на первой строке, то в массив JC записывается номер текущего элемента.

1.3.2 Удаление из матрицы элементов, меньших либо равных q

Для получения из верхнетреугольной матрицы набора матриц, из которых исключены значения, меньшие или равные q, где q — заданное значение, необходимо в цикле по ненулевым элементам матрицы на каждой итерации удалить элемент, если он меньшие или равен q. Для удаления

элемента из матрицы нужно удалить его из массива AN, хранящего значения ненулевых элементов. Затем в массивах NR и NC изменить следующее: тот, кто ссылался на удаляемый элемент, теперь должен ссылаться на элемент, на который ссылался удаляемый. Значения номеров элементов, большие удаляемого, в массивах NR и NC должны уменьшиться на 1. В массивах JR и JC в случае, когда удаляется последний элемент в строке (столбце) записать 0 в позицию массива JR (JC), равную номеру строки. В остальных случаях, когда удаляемый элемент является первым в строке (столбце) и в этой строке (столбце) есть другие элементы, надо вместо номера удаляемого элемента в массив JR (JC) записать номер элемента, на который ссылается удаляемый элемент в массиве NR (NC). Значения номеров элементов, большие удаляемого, в массивах JR и JC также должны уменьшиться на 1.

1.3.3 Вывод матрицы в файл

Вывод матрицы в файл будет происходить в следующем порядке.

1. Вывод массива AN.
2. Вывод массива NR.
3. Вывод массива NC.
4. Вывод массива JR.
5. Вывод массива JC.

1.4 Параллельный алгоритм

При помощи многопоточности можно ускорить процесс обработки матриц. В этом случае для каждой ленты создается отдельный поток. Извлечение и добавление матриц осуществляется потоками. Так, в параллельном алгоритме работа распределяется следующим образом:

— первый поток извлекает матрицу из первой очереди, она обрабатывается на первой ленте, и поток добавляет ее во вторую очередь;

— второй поток извлекает матрицу из второй очереди, она обрабатывается на второй ленте, и поток добавляет ее в третью очередь;

— третий поток извлекает матрицу из второй очереди, и она обрабатывается на третьей ленте;

— каждый поток по завершении вновь извлекает матрицу из своей очереди, и цикл обработки повторяется в параллельном режиме.

Вывод

В данном разделе были описаны основные положения КРМ-схемы хранения разреженных матриц, последовательный и конвейерный алгоритмы обработки матриц.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы вышеизложенных алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема последовательного алгоритма обработки матриц.

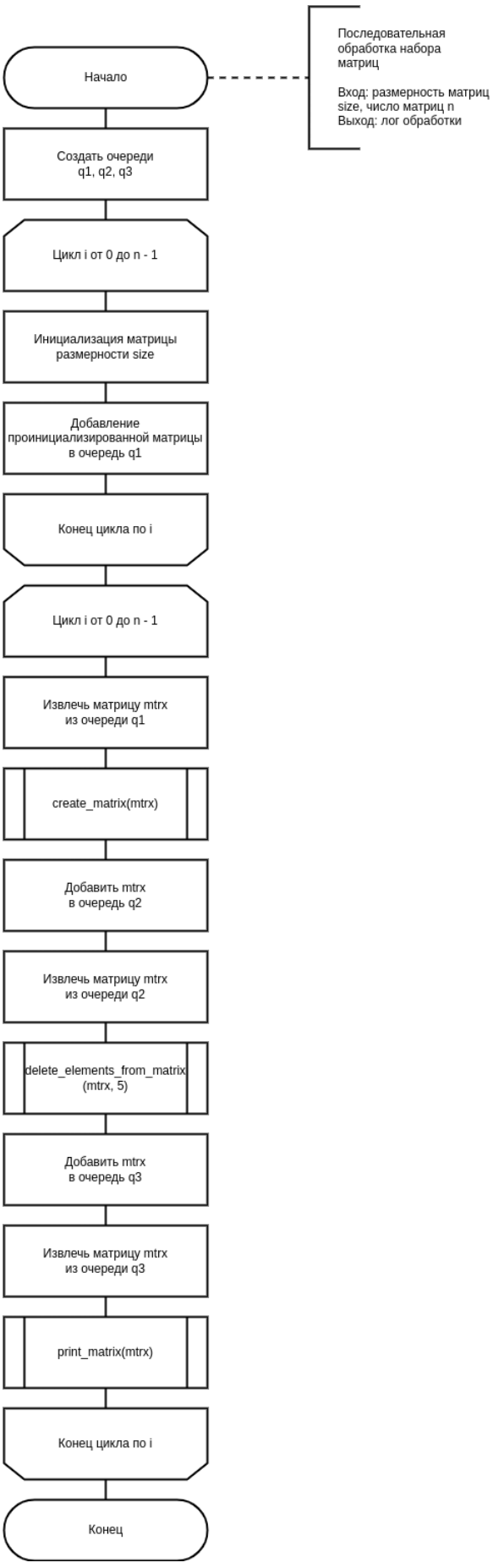

Рисунок 2.1 – Последовательный алгоритм обработки матриц

На рисунке 2.2 представлена схема конвейерного алгоритма обработки матриц.

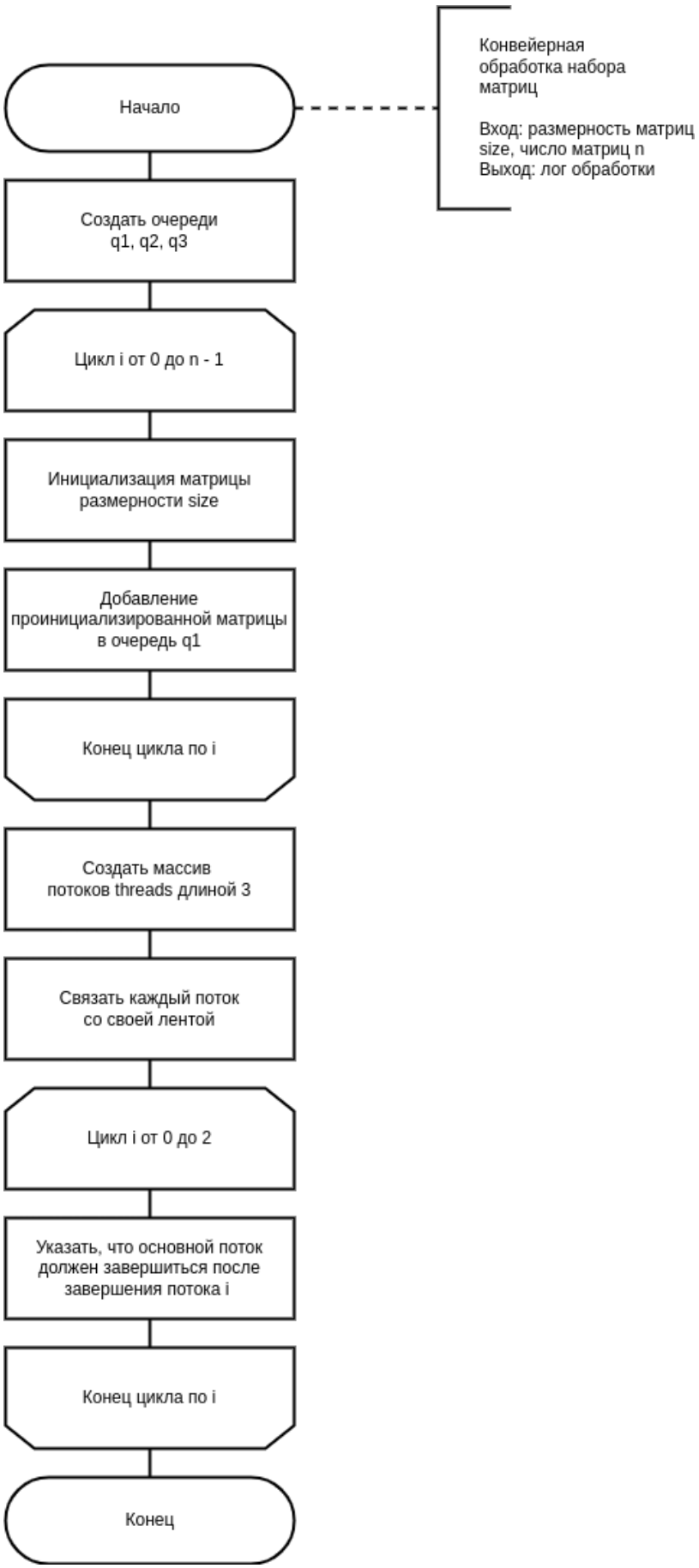

Рисунок 2.2 – Конвейерный алгоритм обработки матриц

На рисунках 2.3–2.5 представлены схемы алгоритмов работы каждой ленты конвейера.

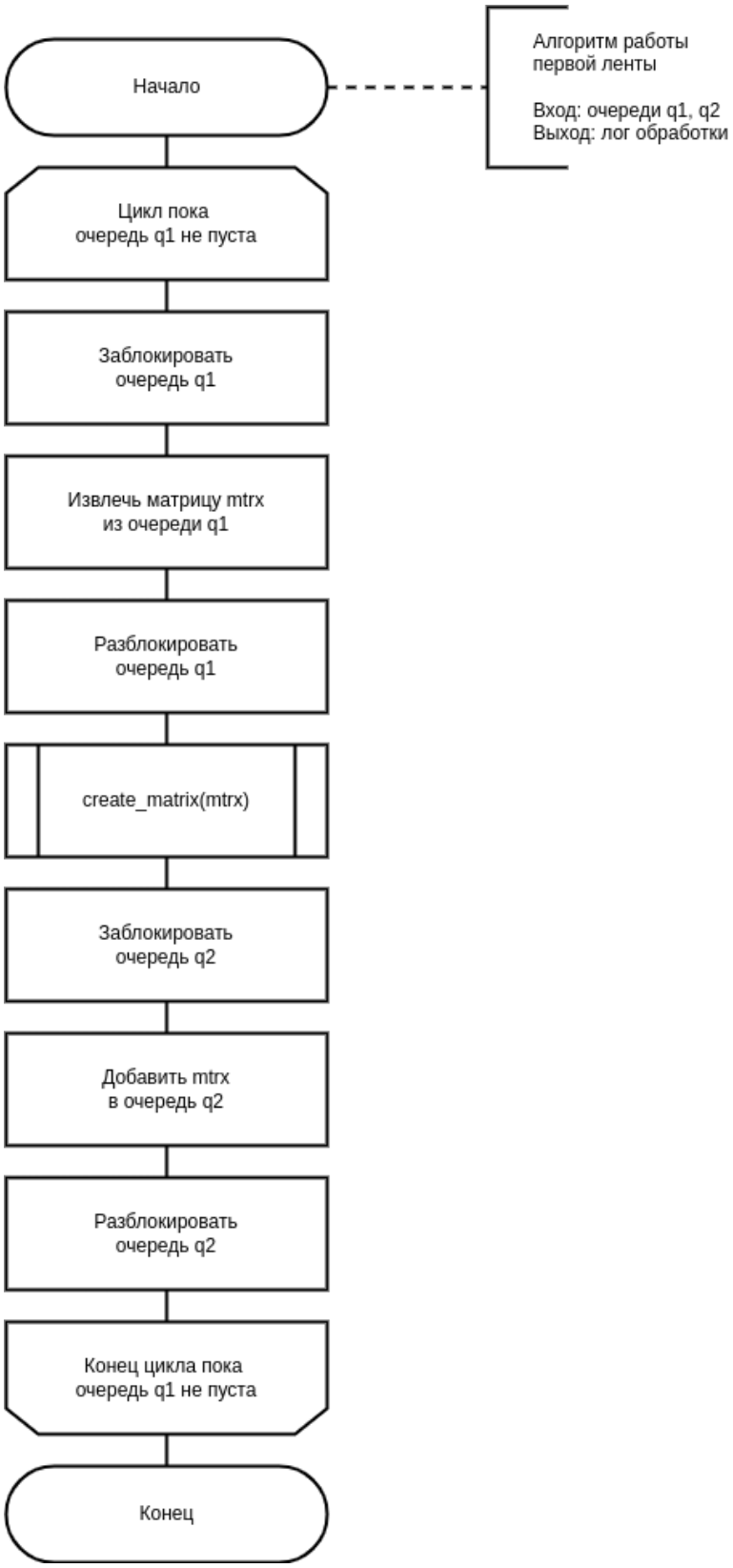

Рисунок 2.3 – Алгоритм работы первой ленты

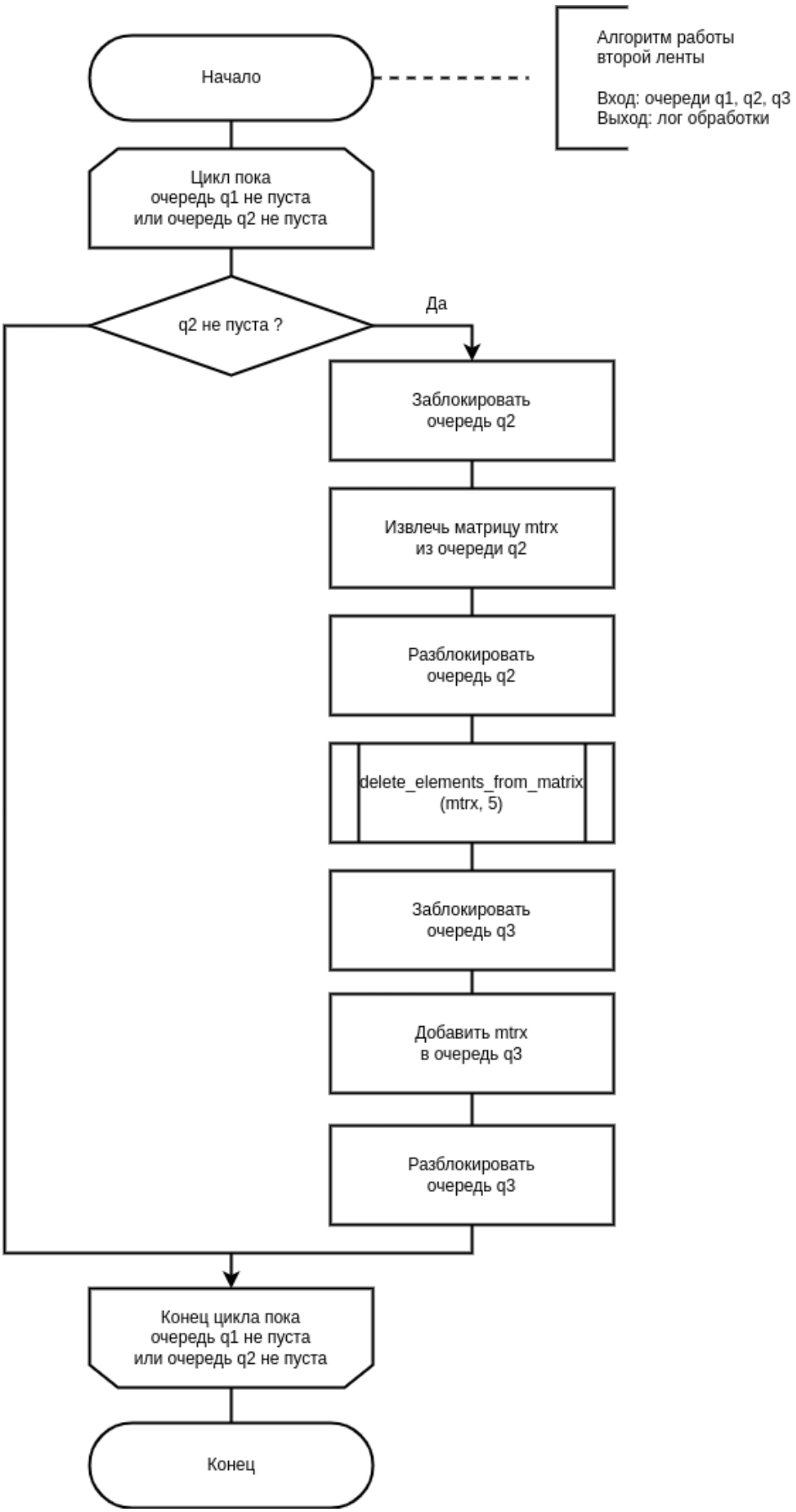

Рисунок 2.4 – Алгоритм работы второй ленты

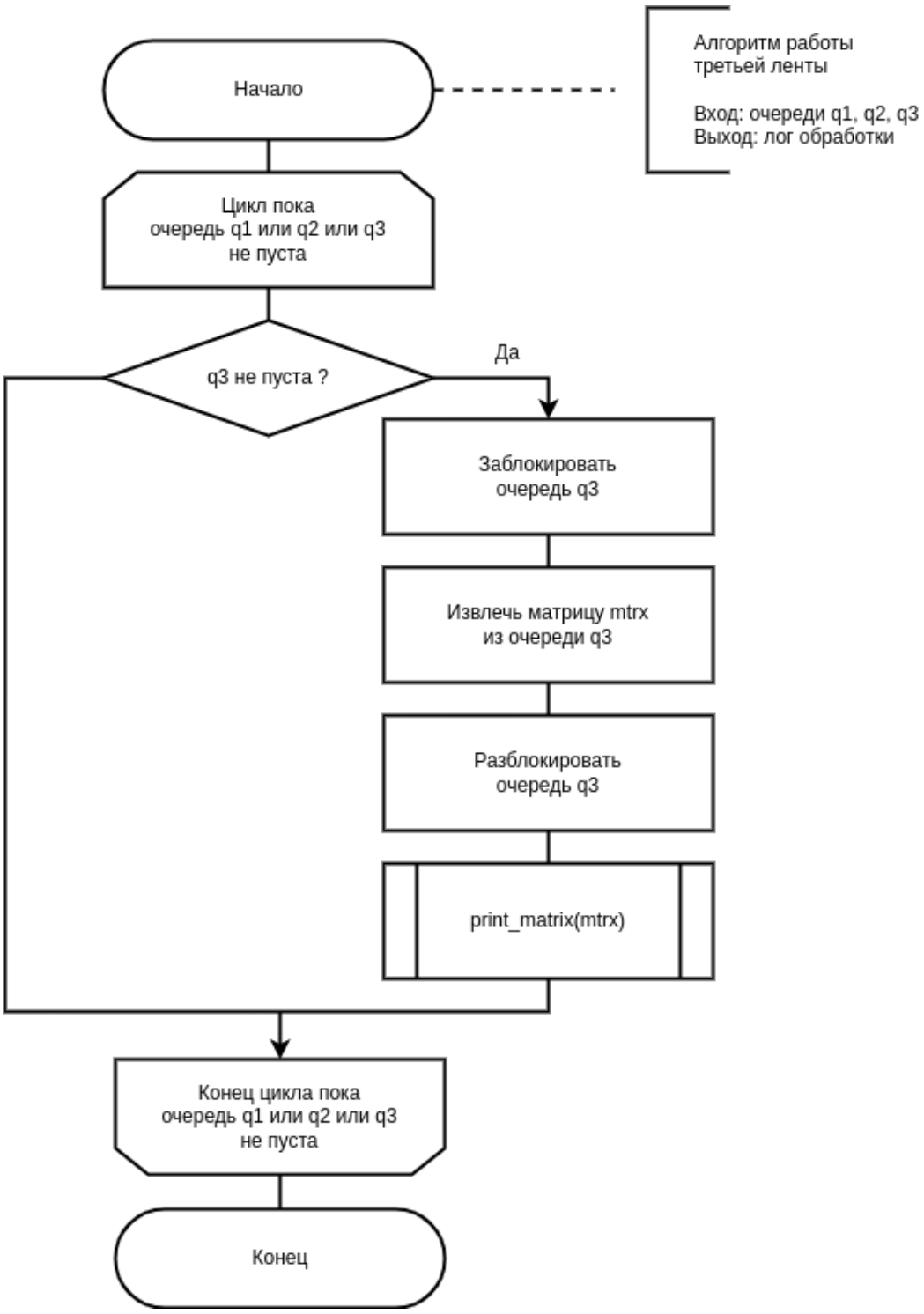

Рисунок 2.5 – Алгоритм работы третьей ленты

На рисунке 2.6 представлена схема алгоритма генерации верхнетреугольной упакованной в КРМ-схему хранения матрицы.

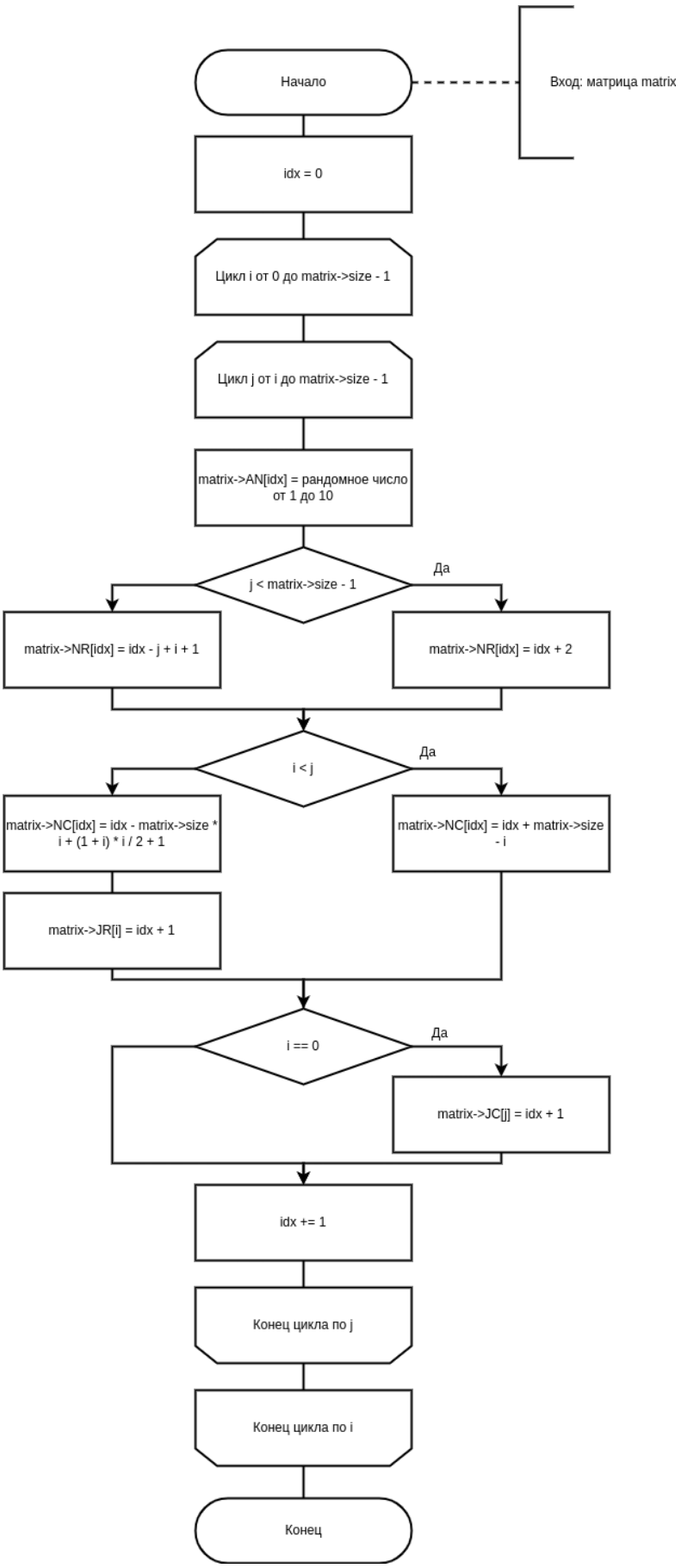

Рисунок 2.6 – Схема алгоритма генерации матрицы

На рисунках 2.7–2.9 представлена схема алгоритма удаления из матрицы элементов, меньших или равных q.

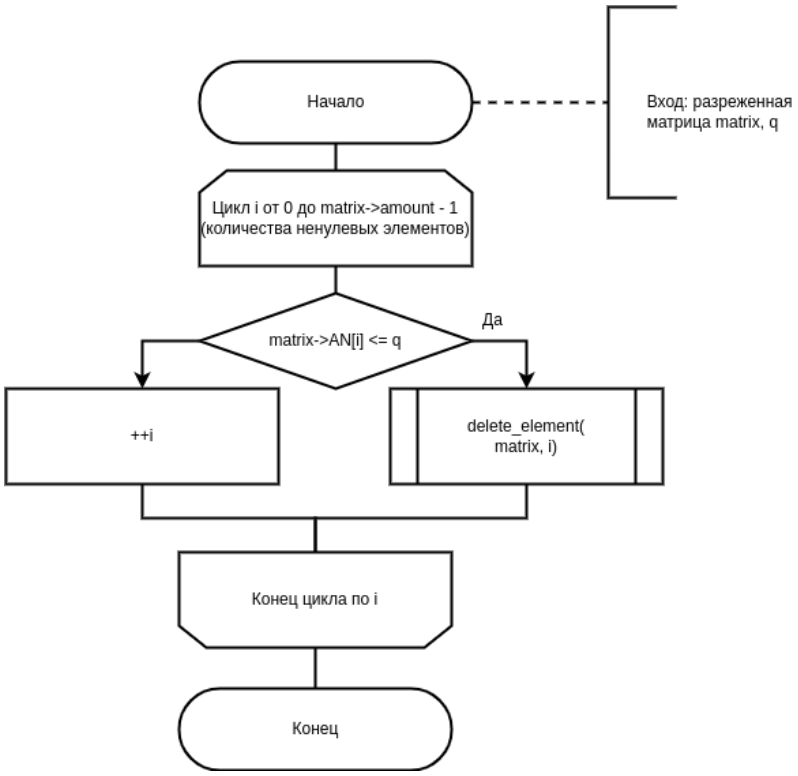

Рисунок 2.7 – Схема алгоритма удаления элементов (1 часть)

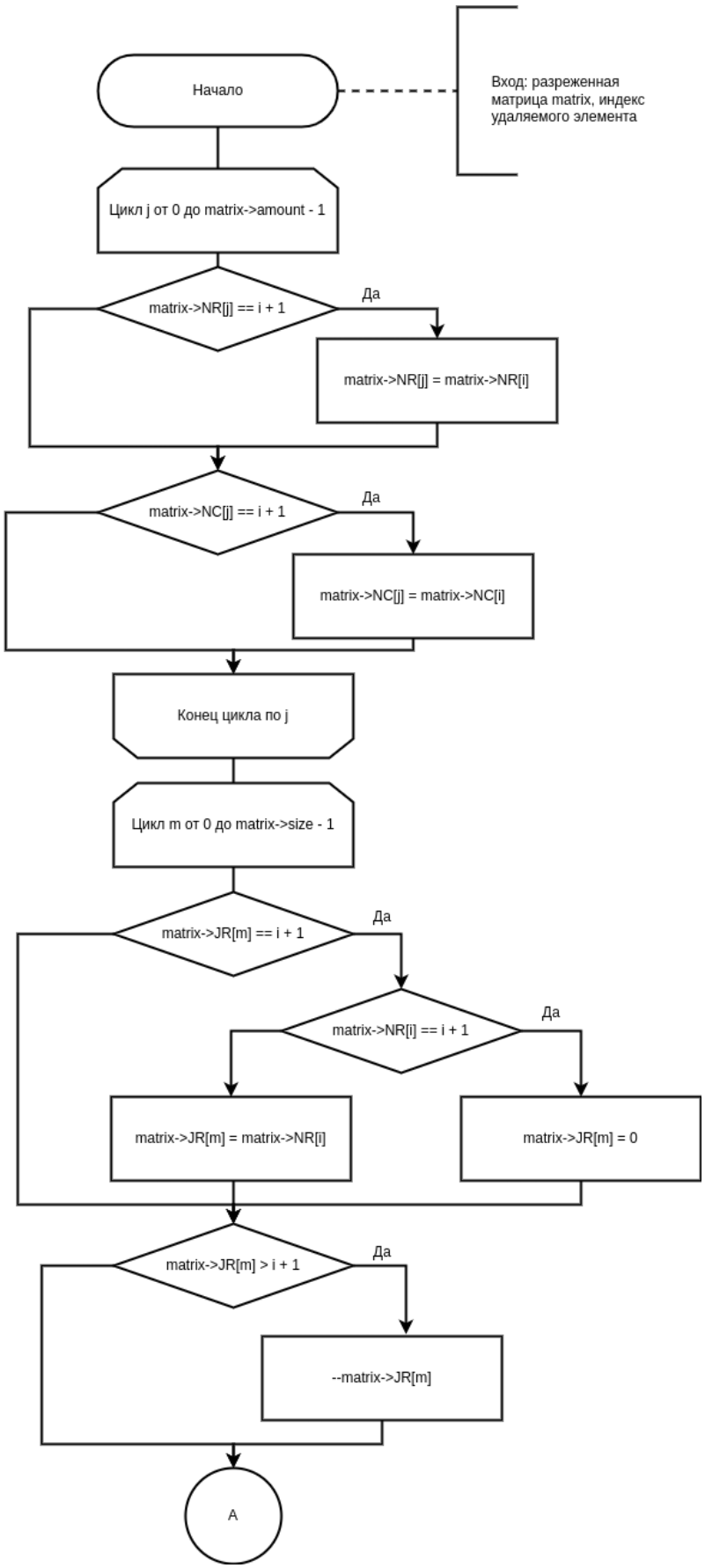

Рисунок 2.8 – Схема алгоритма удаления элементов (2 часть)

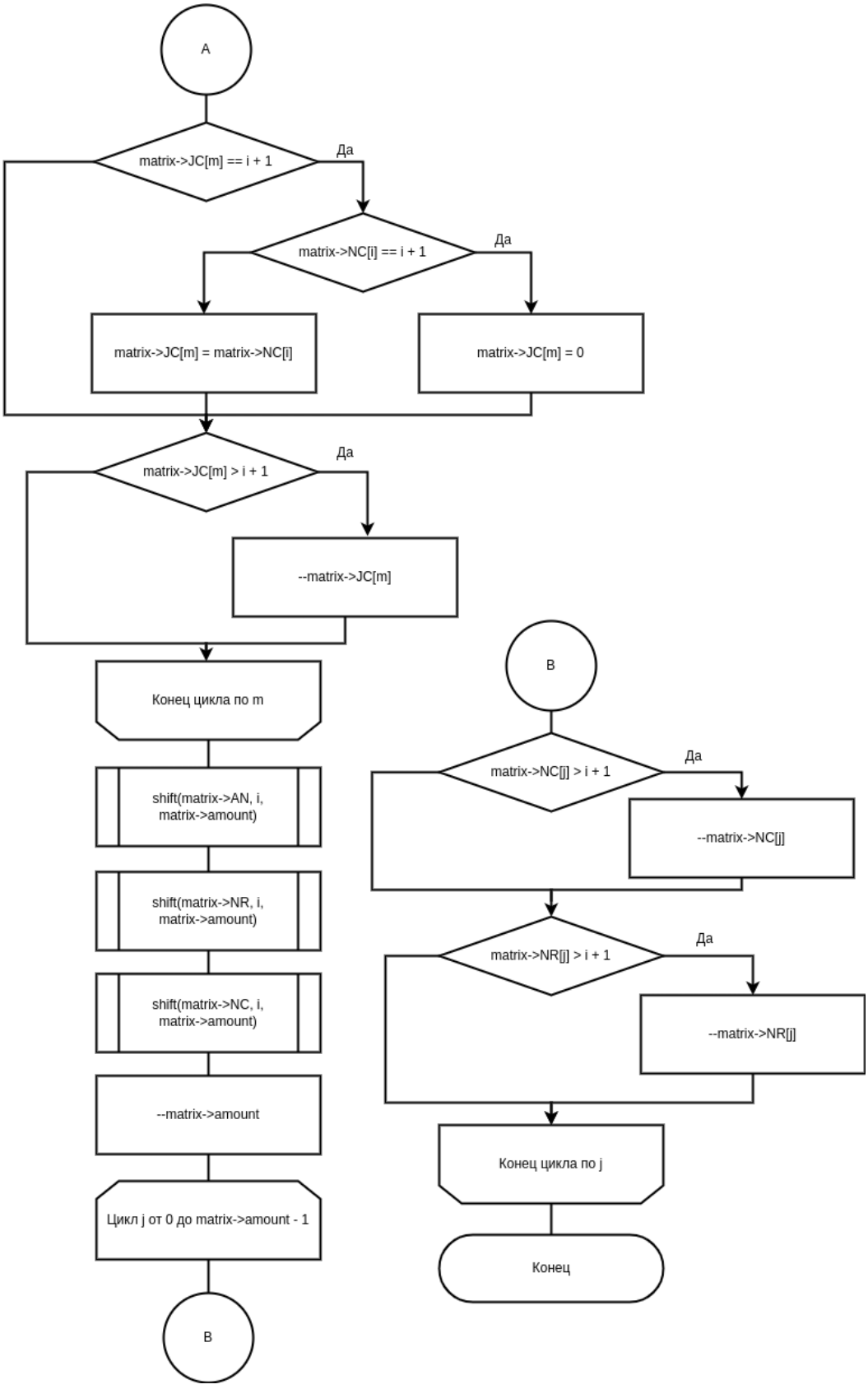

Рисунок 2.9 – Схема алгоритма удаления элементов (3 часть)

На рисунке 2.10 представлена схема алгоритма вывода матрицы в файл.

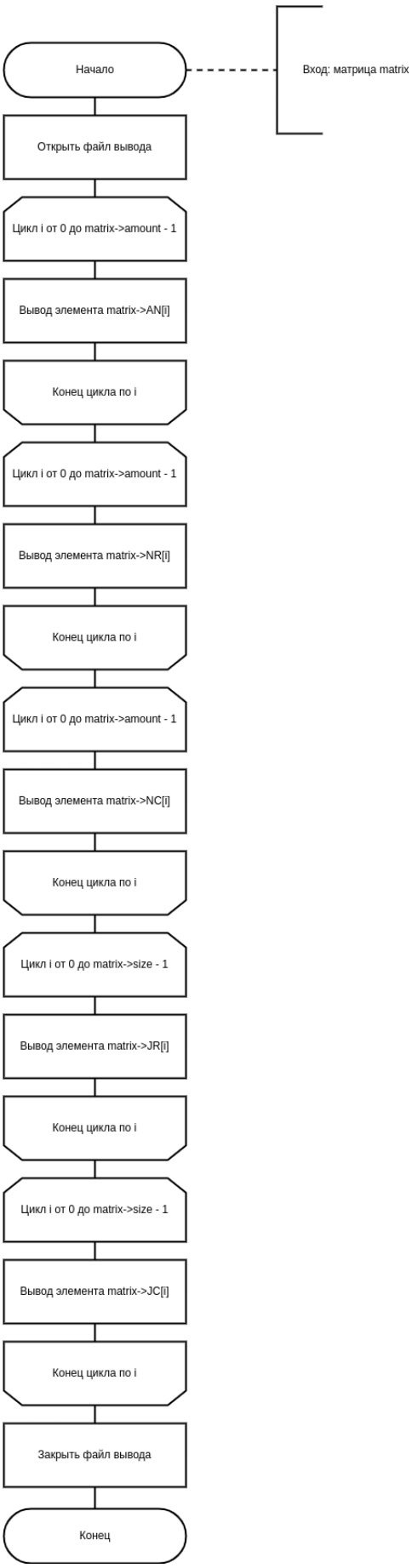

Рисунок 2.10 – Схема алгоритма вывода матрицы в файл

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов.

3 Технологическая часть

В данном разделе приведены средства реализации, сведения о модулях программы, листинги кода, тесты.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык $C++$ [4]. Данный выбор обусловлен наличием инструментов для реализации принципов многопоточного программирования.

Замеры времени проводились при помощи функции $std :: chrono :: system\_clock :: now(...)$ из библиотеки $chrono$ [5].

Реализация графического представления замеров времени производилась при помощи языка программирования $Python$ [6], так как данный язык программирования представляет графическую библиотеку для визуализации данных.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— main.cpp — файл, содержащий точку входа в программу, в нем происходит вызов алгоритмов;

— matrix.cpp и matrix.h — файлы, содержащие алгоритмы работы с матрицами;

— constants.h — файл, содержащий необходимые константы и объявления;

— conveyor.cpp и conveyor.h — файлы, содержащие алгоритмы работы конвейера;

— measurements.cpp и measurements.h — файлы, содержащий функции замеров времени работы алгоритмов;

3.3 Реализация алгоритмов

В листингах 3.1–3.2 представлены реализации последовательного и конвейерного алгоритмов обработки матриц. В листингах 3.3–3.5 представлены алгоритмы работы с матрицами, выполняемые на отдельных лентах конвейера.

Листинг 3.1 – Реализация последовательного алгоритма обработки матриц

```
1
2  void linear_log(matrix_t *mtrx, const int belt, const int task)
3  {
4      std::chrono::time_point<std::chrono::system_clock> start, end;
5      double result = 0;
6
7      if ( belt == FIRST )
8      {
9          start = std::chrono::system_clock::now();
10         create_matrix(mtrx);
11         end = std::chrono::system_clock::now();
12         print_matrix(mtrx, task);
13     }
14     else if ( belt == SECOND )
15     {
16         start = std::chrono::system_clock::now();
17         delete_elements_from_matrix(mtrx, 5);
18         end = std::chrono::system_clock::now();
19     }
20     else if ( belt == THIRD )
21     {
22         start = std::chrono::system_clock::now();
23         print_matrix(mtrx, task);
24         end = std::chrono::system_clock::now();
25         free_matrix(mtrx);
26     }
27
28     result=(std::chrono::duration_cast
          <std::chrono::nanoseconds>(end - start).count()) / IN_SEC;
29     printf("Tape: %d  Task: %d  Start Time: %.6f  End Time:
          %.6f\n", belt, task, cur_time, cur_time + result);
30     cur_time += result;
31 }
32
33
34 void linear_mode(int size, int n)
35 {
36     std::queue<matrix_t> q1;
37     std::queue<matrix_t> q2;
```

```
38      std::queue<matrix_t> q3;
39
40      for ( int i = 0; i < n; ++i )
41      {
42          matrix_t mtrx;
43          init_matrix(&mtrx);
44
45          mtrx.size = size;
46
47          q1.push(mtrx);
48      }
49
50      for ( int i = 0; i < n; ++i )
51      {
52          matrix_t mtrx = q1.front();
53          linear_log(&mtrx, 1, i + 1);
54          q2.push(mtrx);
55          q1.pop();
56
57          mtrx = q2.front();
58          linear_log(&mtrx, 2, i + 1);
59          q3.push(mtrx);
60          q2.pop();
61
62          mtrx = q3.front();
63          linear_log(&mtrx, 3, i + 1);
64          q3.pop();
65      }
66  }
```

Листинг 3.2 – Реализация конвейерного алгоритма обработки матриц

```
1
2   void parallel_log(matrix_t *mtrx, const int belt, const int task)
3   {
4       std::chrono::time_point<std::chrono::system_clock> start, end;
5       double result = 0, start_time;
6
7       if ( belt == FIRST )
8       {
9           start = std::chrono::system_clock::now();
10          create_matrix(mtrx);
```

```
11          end = std::chrono::system_clock::now();
12          result = (std::chrono::duration_cast
              <std::chrono::nanoseconds>(end - start).count()) /
              IN_SEC;
13
14          start_time = cur_time1[task - 1];
15          cur_time1[task] = start_time + result;
16          cur_time2[task - 1] = cur_time1[task];
17          print_matrix(mtrx, task);
18      }
19      else if ( belt == SECOND )
20      {
21          start = std::chrono::system_clock::now();
22          delete_elements_from_matrix(mtrx, 5);
23          end = std::chrono::system_clock::now();
24          result = (std::chrono::duration_cast
              <std::chrono::nanoseconds>(end - start).count()) /
              IN_SEC;
25
26          start_time = cur_time2[task - 1];
27          cur_time2[task] = start_time + result;
28          cur_time3[task - 1] = cur_time2[task];
29      }
30      else if ( belt == THIRD )
31      {
32          start = std::chrono::system_clock::now();
33          print_matrix(mtrx, task);
34          end = std::chrono::system_clock::now();
35          result = (std::chrono::duration_cast
              <std::chrono::nanoseconds>(end - start).count()) /
              IN_SEC;
36
37          start_time = cur_time3[task - 1];
38          cur_time3[task] = start_time + result;
39          free_matrix(mtrx);
40      }
41  }
42
43  void first_belt(std::queue<matrix_t>& q1, std::queue<matrix_t>&
        q2)
44  {
```

```
45      int task = 0;
46      std::mutex m;
47
48      while ( !q1.empty())
49      {
50          m.lock();
51          matrix_t mtrx = q1.front();
52          m.unlock();
53
54          parallel_log(&mtrx, 1, ++task);
55
56          m.lock();
57          q2.push(mtrx);
58          q1.pop();
59          m.unlock();
60      }
61  }
62
63
64  void second_belt(std::queue<matrix_t>& q1, std::queue<matrix_t>&
        q2, std::queue<matrix_t>& q3)
65  {
66      int task = 0;
67      std::mutex m;
68
69      do
70      {
71          if ( !q2.empty() )
72          {
73              m.lock();
74              matrix_t mtrx = q2.front();
75              m.unlock();
76
77              parallel_log(&mtrx, 2, ++task);
78
79              m.lock();
80              q3.push(mtrx);
81              q2.pop();
82              m.unlock();
83          }
84      } while ( !q1.empty() || !q2.empty() );
```

```
85 }
86
87
88 void third_belt(std::queue<matrix_t>& q1, std::queue<matrix_t>&
       q2, std::queue<matrix_t>& q3)
89 {
90     int task = 0;
91     std::mutex m;
92
93     do
94     {
95         if ( !q3.empty() )
96         {
97             m.lock();
98             matrix_t mtrx = q3.front();
99             m.unlock();
100
101             parallel_log(&mtrx, 3, ++task);
102
103             m.lock();
104             q3.pop();
105             m.unlock();
106         }
107     } while ( !q1.empty() || !q2.empty() || !q3.empty() );
108 }
109
110
111 void parallel_mode(int size, int n)
112 {
113     std::queue<matrix_t> q1;
114     std::queue<matrix_t> q2;
115     std::queue<matrix_t> q3;
116
117     for ( int i = 0; i < n; ++i )
118     {
119         matrix_t mtrx;
120         init_matrix(&mtrx);
121
122         mtrx.size = size;
123
124         q1.push(mtrx);
```

```
125     }
126
127     for ( int i = 0; i < n + 1; ++i )
128     {
129         cur_time1.push_back(0);
130         cur_time2.push_back(0);
131         cur_time3.push_back(0);
132     }
133
134     std::vector<std::thread> threads(COUNT_THREADS);
135
136     threads[0] = std::thread(first_belt, std::ref(q1),
            std::ref(q2));
137     threads[1] = std::thread(second_belt, std::ref(q1),
            std::ref(q2), std::ref(q3));
138     threads[2] = std::thread(third_belt, std::ref(q1),
            std::ref(q2), std::ref(q3));
139
140     for ( int i = 0; i < COUNT_THREADS; ++i )
141     {
142         threads[i].join();
143     }
144
145     cout << endl;
146     for ( int i = 0; i < n; ++i )
147     {
148         printf("Tape: 1   Task: %d   Start Time: %.6f   End Time:
                %.6f\n", i + 1, cur_time1[i], cur_time1[i + 1]);
149     }
150     for ( int i = 0; i < n; ++i )
151     {
152         printf("Tape: 2   Task: %d   Start Time: %.6f   End Time:
                %.6f\n", i + 1, cur_time2[i], cur_time2[i + 1]);
153     }
154     for ( int i = 0; i < n; ++i )
155     {
156         printf("Tape: 3   Task: %d   Start Time: %.6f   End Time:
                %.6f\n", i + 1, cur_time3[i], cur_time3[i + 1]);
157     }
158 }
```

Листинг 3.3 – Алгоритм генерации верхнетреугольной упакованной в КРМ-схему хранения матрицы

```
1
2  void create_matrix(matrix_t* matrix)
3  {
4      srand(time(0));
5
6      matrix->amount = (1 + matrix->size) * matrix->size / 2;
7
8      matrix->JR = new int[matrix->size];
9      matrix->JC = new int[matrix->size];
10     matrix->AN = new int[matrix->amount];
11     matrix->NR = new int[matrix->amount];
12     matrix->NC = new int[matrix->amount];
13
14     int idx = 0;
15     for (int i = 0; i < matrix->size; ++i) {
16         for (int j = i; j < matrix->size; ++j) {
17             int tmp;
18             tmp = rand() % MAX_NUM + 1;
19
20             matrix->AN[idx] = tmp;
21             if (j < matrix->size - 1) {
22                 matrix->NR[idx] = idx + 2;
23             } else {
24                 matrix->NR[idx] = idx - j + i + 1;
25             }
26             if (i < j) {
27                 matrix->NC[idx] = idx + matrix->size - i;
28             } else {
29                 matrix->NC[idx] = idx - matrix->size * i + (1 +
                       i) * i / 2 + 1;
30                 matrix->JR[i] = idx + 1;
31             }
32
33             if (i == 0) {
34                 matrix->JC[j] = idx + 1;
35             }
36             ++idx;
37         }
38     }
```

```
39 }
```

Листинг 3.4 – Алгоритм удаления элементов из матрицы меньших или равных q

```
1
2  void delete_element(int *arr, int idx, int n) {
3      for (int i = idx; i < n - 1; ++i) {
4          arr[i] = arr[i + 1];
5      }
6  }
7
8  void delete_elements_from_matrix(matrix_t *matrix, int q) {
9      for (int i = 0; i < matrix->amount;) {
10         if (matrix->AN[i] <= q) {
11             for (int j = 0; j < matrix->amount; ++j) {
12                 if (matrix->NR[j] == i + 1) {
13                     matrix->NR[j] = matrix->NR[i];
14                 }
15                 if (matrix->NC[j] == i + 1) {
16                     matrix->NC[j] = matrix->NC[i];
17                 }
18             }
19             for (int m = 0; m < matrix->size; ++m) {
20                 if (matrix->JR[m] == i + 1) {
21                     if (matrix->NR[i] == i + 1) {
22                         matrix->JR[m] = 0;
23                     } else {
24                         matrix->JR[m] = matrix->NR[i];
25                     }
26                 }
27                 if (matrix->JR[m] > i + 1) {
28                     --matrix->JR[m];
29                 }
30                 if (matrix->JC[m] == i + 1) {
31                     if (matrix->NC[i] == i + 1) {
32                         matrix->JC[m] = 0;
33                     } else {
34                         matrix->JC[m] = matrix->NC[i];
35                     }
36                 }
37                 if (matrix->JC[m] > i + 1) {
```

```
38                      --matrix->JC[m];
39                  }
40              }
41              delete_element(matrix->NR, i, matrix->amount);
42              delete_element(matrix->AN, i, matrix->amount);
43              delete_element(matrix->NC, i, matrix->amount);
44              --matrix->amount;
45              for (int j = 0; j < matrix->amount; ++j) {
46                  if (matrix->NC[j] > i + 1) {
47                      --matrix->NC[j];
48                  }
49                  if (matrix->NR[j] > i + 1) {
50                      --matrix->NR[j];
51                  }
52              }
53          } else {
54              ++i;
55          }
56      }
57  }
```

Листинг 3.5 – Алгоритм вывода матрицы в файл

```
1
2           void print_matrix(matrix_t* matrix, int num)
3           {
4               ofstream fout("matrix" + to_string(num) + ".txt",
                    ios_base::app);
5               fout << "AN:";
6               for (int i = 0; i < matrix->amount; ++i) {
7                   fout << matrix->AN[i] << "␣";
8               }
9               fout << endl;
10
11              fout << "NR:";
12              for (int i = 0; i < matrix->amount; ++i) {
13                  fout << matrix->NR[i] << "␣";
14              }
15              fout << endl;
16
17              fout << "NC:";
18              for (int i = 0; i < matrix->amount; ++i) {
```

```
19                  fout << matrix->NC[i] << " ";
20              }
21              fout << endl;
22
23              fout << "JR:";
24              for (int i = 0; i < matrix->size; ++i) {
25                  fout << matrix->JR[i] << " ";
26              }
27              fout << endl;
28
29              fout << "JC:";
30              for (int i = 0; i < matrix->size; ++i) {
31                  fout << matrix->JC[i] << " ";
32              }
33              fout << endl;
34              fout << endl;
35              fout << endl;
36
37              fout.close();
38          }
```

3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для реализованных алгоритмов.

Таблица 3.1 – Функциональные тесты

Входные матрицы ($k = 5$, матрица одна)	Ожидаемый результат
[]	[]
AN:2 9 6 1 2 7 1 2 10 3 3 6 5 7 6 NR:2 3 4 5 1 7 8 9 6 11 12 10 14 13 15 NC:1 6 7 8 9 2 10 11 12 3 13 14 4 15 5 JR:1 6 10 13 15 JC:1 2 3 4 5	AN:9 6 7 10 6 7 6 NR:2 1 4 3 5 6 7 NC:3 2 1 5 6 7 4 JR:1 3 5 6 7 JC:0 1 2 0 4

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функци-

ональное тестирование пройдено успешно реализациями двух алгоритмов обработки — последовательной и конвейерной.

Вывод

Были реализованы последовательный и конвейерный алгоритмы обработки матриц.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализаций алгоритма по затраченному процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование, приведены ниже:

— операционная система Manjaro Linux [7];

— память 7,6 ГБ;

— процессор 8 × Intel® Core™ i5-10210U CPU @ 1.60 ГГц [8] с 4 физическими ядрами и 8 логическими.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы. На этом рисунке пользователь выбирает из меню пункт 2 — конвейерную реализацию алгоритма, вводит размерность и количество матриц и получает на выходе отметки времени начала и конца обработки заявки.

```
МЕНЮ:
1. Последовательная обработка
2. Конвейерная обработка
3. Замер времени работы алгоритма в зависимости от количества матриц
0. Выход
Выбор: 2
Введите размерность матриц:
5
Введите количество матриц:
5

Tape: 1  Task: 1  Start Time: 0.000000  End Time: 0.000083
Tape: 1  Task: 2  Start Time: 0.000083  End Time: 0.000091
Tape: 1  Task: 3  Start Time: 0.000091  End Time: 0.000107
Tape: 1  Task: 4  Start Time: 0.000107  End Time: 0.000115
Tape: 1  Task: 5  Start Time: 0.000115  End Time: 0.000121
Tape: 2  Task: 1  Start Time: 0.000083  End Time: 0.000103
Tape: 2  Task: 2  Start Time: 0.000103  End Time: 0.000107
Tape: 2  Task: 3  Start Time: 0.000107  End Time: 0.000121
Tape: 2  Task: 4  Start Time: 0.000121  End Time: 0.000131
Tape: 2  Task: 5  Start Time: 0.000131  End Time: 0.000140
Tape: 3  Task: 1  Start Time: 0.000103  End Time: 0.000379
Tape: 3  Task: 2  Start Time: 0.000379  End Time: 0.000506
Tape: 3  Task: 3  Start Time: 0.000506  End Time: 0.000622
Tape: 3  Task: 4  Start Time: 0.000622  End Time: 0.000711
Tape: 3  Task: 5  Start Time: 0.000711  End Time: 0.000899
```

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Функция $std :: chrono :: system\_clock :: now(...)$ из библиотеки *chrono* языка программирования $C++$ возвращает процессорное время в секундах — значение типа float.

На рисунке 4.2 приведены результаты замеров времени выполнения однопоточной и многопоточной реализаций заданного алгоритма.

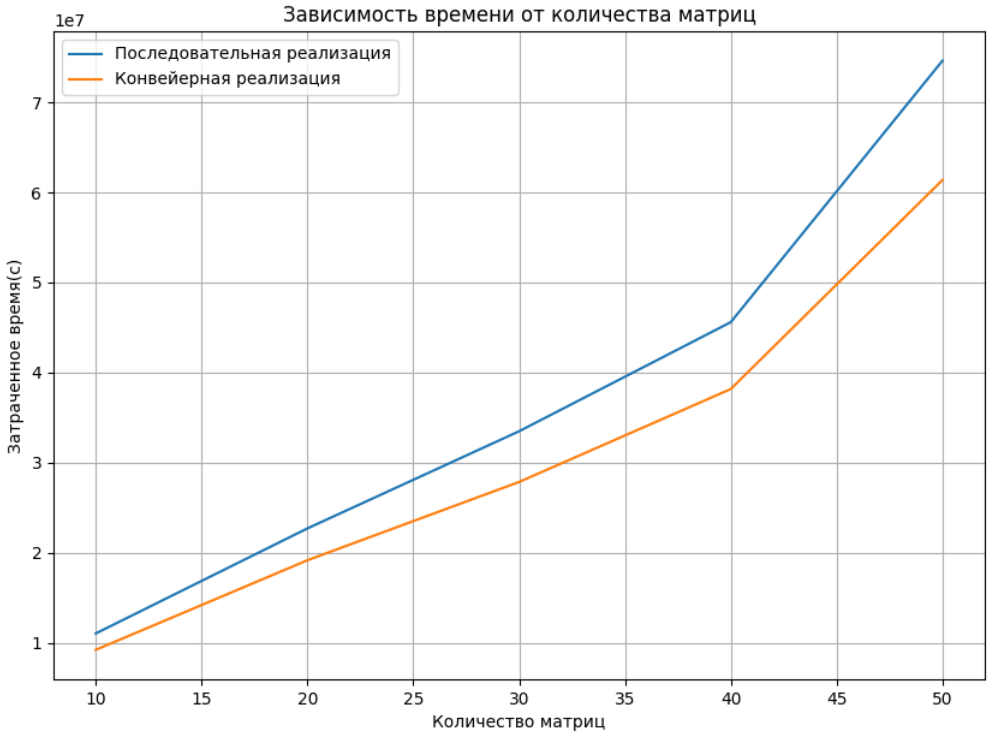

Рисунок 4.2 – Сравнение времени выполнения реализаций алгоритмов

4.4 Вывод

Приведенные характеристики времени показывают, что конвейерная реализация выигрывает по времени у последовательной реализации.

Заключение

В результате выполнения данной лабораторной работы были рассмотрены последовательный и конвейерный алгоритмы обработки разреженных матриц, упакованных по КРМ-схеме, построены схемы, соответствующие данным алгоритмам, и выполнена их реализация. Проведенные замеры времени работы реализаций показали, что конвейерная реализация работает быстрее последовательной.

В рамках выполнения работы цель достигнута: получен навык организации асинхронного взаимодействия между потоками на примере моделирования конвейера обработки разреженных матриц, представленных с помощью кольцевой КРМ-схемы.

Решены все задачи:

— описана КРМ-схема хранения матриц;

— разработан и реализован последовательный алгоритм;

— выделены алгоритмы, выполняемые на отдельных лентах конвейера;

— разработана и реализована конвейерная версия данного алгоритма;

— созданы схемы изучаемых алгоритмов;

— определены средства программной реализации;

— выполнены замеры процессорного времени работы реализаций алгоритма;

— проведен сравнительный анализ по времени работы реализаций алгоритмов;

— подготовлен отчет о выполненной лабораторной работе.

Список использованных источников

[1] Параллельная обработка данных [Электронный ресурс]. Режим доступа: https://parallel.ru/vvv/lec1.html (дата обращения: 25.11.2021).

[2] А. В. Силантьева. Лекции по типам и структурам данных. — М.: МГТУ им. Н.Э.Баумана. — 419 с.

[3] С. Писсанецки. Технология разреженных матриц. — М.: Мир, 1988. — 410 с.

[4] Документация по языку C++ [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 25.11.2022).

[5] Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 25.11.2022).

[6] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 25.11.2022).

[7] Manjaro Linux [Электронный ресурс]. Режим доступа: https://manjaro.org (дата обращения: 25.11.2022).

[8] Процессор Intel® Core™ i5-10210U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/195436/intel-core-i510210u-processor-6m-cache-up-to-4-20-ghz.html (дата обращения: 25.11.2022).