

Национальный исследовательский университет

«Высшая школа экономики»

Факультет компьютерных наук

Департамент

Программная инженерия

Самостоятельная работа по дисциплине

«Архитектура вычислительных систем»

Тема работы: Определить ранг матрицы. Входные данные: целое положительное число n , произвольная матрица A размерности $n \times n$. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

Выполнил: студент группы БПИ191(1) Бен Мустафа Анас Риадович.

Преподаватель: Легалов Александр Иванович

Москва 2020

Вариант 5

Определить ранг матрицы. Входные данные: целое положительное число n , произвольная матрица A размерности $n \times n$. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

1. Описание принципа построения работы программы.

Для решения данной задачи использовалась модель построения многопоточных приложений, называемая **итеративный параллелизм**, используемый для реализации нескольких потоков (часто идентичных), каждый из которых содержит циклы. Потоки программы, описываются итеративными функциями и работают совместно над решением одной задачи.

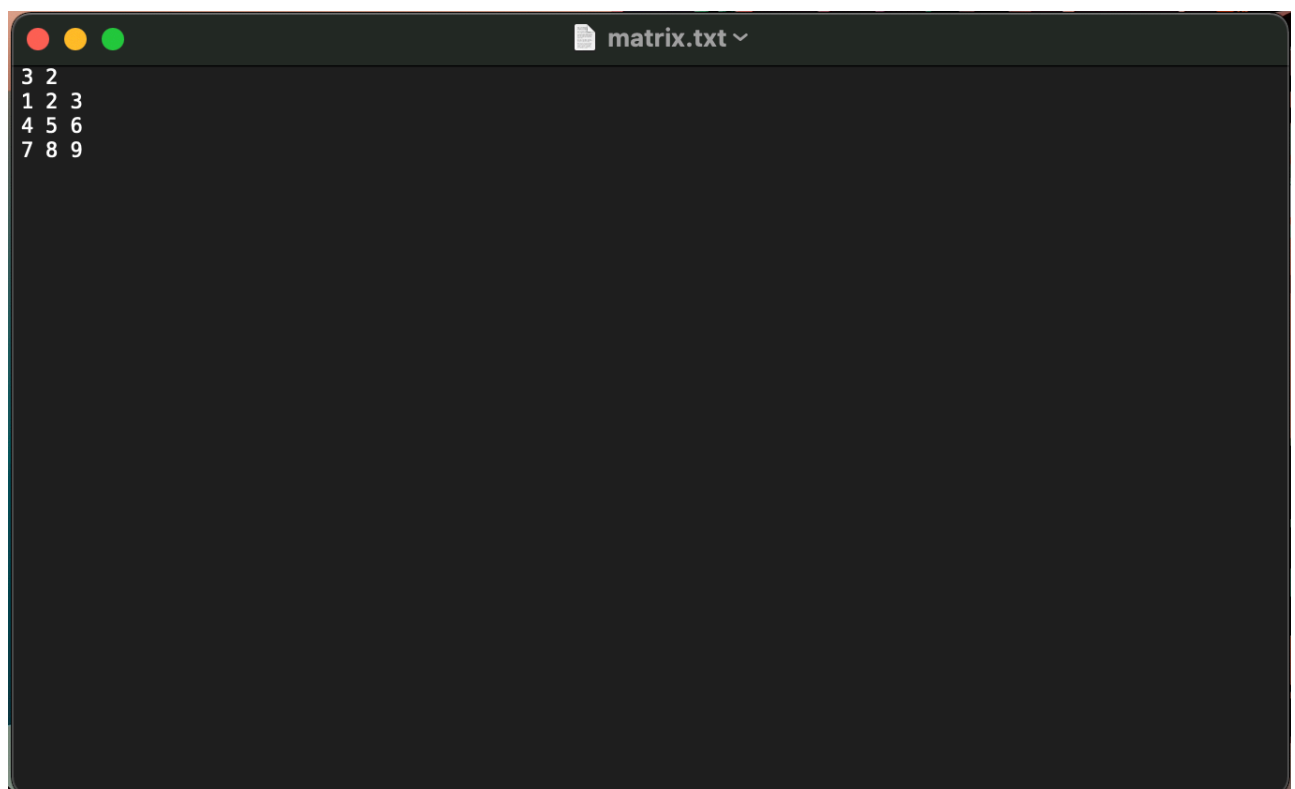
Вычисления ранга матрицы были проведены с помощью модифицированной версии метода Гаусса (наиболее простой способ с точки зрения программной реализации).

Информация по алгоритму взята с ресурса: https://studopedia.ru/21_129494_modifitsirovanniy-metod-gaussa.html

2. Описание входных данных.

Для удобства использования программы входные данные могут быть представлены в двух видах:

1. Данные могут быть переданы в программу через текстовый файл в следующем формате, где левый параметр первой строки - размерность матрицы, правый параметр первой строки - количество используемых потоков. Затем матрица размера $n \times n$.



```
3 2
1 2 3
4 5 6
7 8 9
```

2. Данные могут быть переданы в программу пользователем вручную.

```
/Users/anasbenmustafa/CLionProjects/Rank_HW4/cmake-build-debug/Rank_HW4
```

```
Введите n (размерность матрицы) : 3
```

```
Введите количество потоков : 2
```

```
Введите элемент (0;0) : 1
```

```
Введите элемент (0;1) : 2
```

```
Введите элемент (0;2) : 3
```

```
Введите элемент (1;0) : 4
```

```
Введите элемент (1;1) : 5
```

```
Введите элемент (1;2) : 6
```

```
Введите элемент (2;0) : 7
```

```
Введите элемент (2;1) : 8
```

```
Введите элемент (2;2) : 9
```

```
Полученная матрица :
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

```
Полученный ранг матрицы: 3
```

```
Время работы программы : 14553 ms
```

3. Описание выходных данных.

Вне зависимости от типа входных данных (вручную или текстовым документом), в качестве выходных данных программы пользователь получает исходную матрицу, посчитанный ранг исходной матрицы, а также итоговое время работы программы. Алгоритм обрабатывает практически мгновенно. (0ms на фото ниже - не ошибка).

```
Полученная матрица :
```

```
1 2 3 4
```

```
2 3 4 5
```

```
3 4 5 6
```

```
5 6 7 8
```

```
Полученный ранг матрицы : 4
```

```
Время работы программы : 0 ms
```

Список используемых источников

1. <https://habr.com/ru/post/443406/>
2. <https://docs.microsoft.com/ru-ru/cpp/standard-library/thread-classview=msvc-160&viewFallbackFrom=vs-2019>
3. https://l.wzm.me/_coder/custom/parallel.programming/001.htm

Полный текст программы

```
1  #include <iostream>
2  #include <vector>
3  #include <fstream>
4  #include <thread>
5  #include <chrono>
6
7  using namespace std;
8
9  int n, quantityOfThreads;
10 thread *threads;
11 vector<bool> usedMatrixLines;
12 int rankOfMatrix;
13 int **matrix;
14 const double epsilon = 1E-9;
15
16 void inputMatrix() {
17     int elem;
18
19     matrix = new int* [n]; // Инициализация матрицы
20     for (int i = 0; i < n; i++) {
21         matrix[i] = new int[n];
22     }
23
24     for (int i = 0; i < n; ++i) {
25         for (int j = 0; j < n; ++j) {
26             cout << "Введите элемент (" << i << " "; << j << ") : ";
27             cin >> elem;
28             matrix[i][j] = elem;
29         }
30     }
31 }
32
33 void computeRank(int temporary) {
34     int tmp = 0;
35
36     for (int i = 0; i < n; ++i) {
37         if (!usedMatrixLines[i] && abs(matrix[i][temporary]) > epsilon) {
38             tmp = i;
39             break;
40         }
41     }
42
43     if (n == tmp) {
44         rankOfMatrix -= 1;
45     } else {
46         usedMatrixLines[tmp] = true;
47     }
48 }
49
50 void changeMatrix(int j) {
51     int tmp = 0;
52
53     for (int i = 0; i < n; ++i) {
54         if (!usedMatrixLines[i] && abs(matrix[i][j]) > epsilon) {
55             tmp = i;
56             break;
57         }
58     }
59
60     if (n != tmp) {
61         for (int m = j + 1; m < n; ++m) {
62             matrix[tmp][m] /= matrix[tmp][j];
63         }
64         for (int q = 0; q < n; ++q) {
65             if (q != tmp && abs(matrix[q][j]) > epsilon) {
66                 for (int r = j + 1; r < n; ++r) {
67                     matrix[q][r] -= matrix[tmp][r] * matrix[q][j];
68                 }
69             }
70         }
71     }
72 }
73
74 void matrixRankFunc() {
```

```

75     if (quantityOfThreads == n) {
76         for (int i = 0; i < n; ++i) {
77             threads[i] = thread(changeMatrix, i);
78         }
79
80         for (int i = 0; i < n; ++i) {
81             threads[i].join();
82             computeRank(i);
83         }
84     } else {
85         int tmp = 0;
86
87         for (int i = 0; i < n; ++i) {
88             if (tmp < quantityOfThreads) {
89                 threads[tmp] = thread(changeMatrix, i);
90                 ++tmp;
91             }
92             if (tmp == quantityOfThreads) {
93                 tmp = 0;
94                 for (int j = 0; j < quantityOfThreads; ++j) {
95                     threads[j].join();
96                 }
97             }
98         }
99
100        for (int i = 0; i < tmp; ++i) {
101            threads[i].join();
102        }
103
104        for (int j = 0; j < n; ++j) {
105            computeRank(j);
106        }
107    }
108 }
109
110 void printMatrix() {
111     cout << endl << "Полученная матрица : " << endl;
112     for (int i = 0; i < n; i++) {
113         for (int j = 0; j < n; j++) {
114             cout << matrix[i][j] << " ";
115         }
116         cout << endl;
117     }
118 }
119
120 void GetMatrixFromFile(string pathToFile) {
121     ifstream in(pathToFile);
122     in >> n >> quantityOfThreads;
123
124     cout << "Полученное значение n : " << n << endl << "Полученное значение кол-ва потоков : " << quantityOfThreads << endl;
125     rankOfMatrix = n;
126     usedMatrixLines = vector<bool>(n);
127
128     matrix = new int *[n];
129
130     for (int i = 0; i < n; i++) {
131         matrix[i] = new int[n];
132     }
133
134     for (int i = 0; i < n; ++i) {
135         for (int j = 0; j < n; ++j) {
136             in >> matrix[i][j];
137         }
138     }
139 }
140
141 int main(int argc, char* argv[]) {
142
143     // Начинаем отсчёт времени со старта работы программы
144     auto begin = std::chrono::steady_clock::now();
145
146     // В случае, если пользователь не передаёт в программу путь к файлу с матрицей, делаем ручной ввод
147     if (argc == 1) {
148         cout << "Введите n (размерность матрицы) : ";

```

```

148     cout << "Введите n (размерность матрицы) : ";
149     cin >> n;
150     cout << "Введите количество потоков : ";
151     cin >> quantityOfThreads;
152
153     if (quantityOfThreads > n || n < 1 || quantityOfThreads < 1) {
154         cout << endl << "Некорректный ввод!" << endl <<
155             "1. Количество потоков должно быть <= размерности матрицы." << endl <<
156             "2. Размерность матрицы должна быть >= 1." << endl <<
157             "3. Количество потоков должно быть >= 1.";
158         return 0;
159     }
160
161     rankOfMatrix = n;
162     threads = new thread[quantityOfThreads];
163     usedMatrixLines = vector<bool>(n);
164
165     inputMatrix(); // ввод пользователем матрицы
166     printMatrix(); // вывод матрицы на экран
167     matrixRankFunc(); // функция подсчета ранга матрицы
168
169     cout << endl << "Полученный ранг матрицы: " << rankOfMatrix << endl; // Вывод результатов работы потока для ранга
170
171     auto end = chrono::steady_clock::now();
172     auto elapsed_ms = chrono::duration_cast<std::chrono::milliseconds>(end - begin);
173     cout << "Время работы программы : " << elapsed_ms.count() << " ms\n";
174 }
175
176 else if (argc == 2) { // Если передан файл с матрицей - считываем его
177
178     GetMatrixFromFile( pathToFile: argv[1]);
179
180     if (quantityOfThreads > n || n < 1 || quantityOfThreads < 1) {
181         cout << endl << "Некорректный ввод!" << endl <<
182             "1. Количество потоков должно быть <= размерности матрицы." << endl <<
183             "2. Размерность матрицы должна быть >= 1." << endl <<
184             "3. Количество потоков должно быть >= 1.";
185         return 0;
186     }
187
188     cout << endl << "Полученный ранг матрицы: " << rankOfMatrix << endl; // Вывод результатов работы потока для ранга
189
190     auto end = chrono::steady_clock::now();
191     auto elapsed_ms = chrono::duration_cast<std::chrono::milliseconds>(end - begin);
192     cout << "Время работы программы : " << elapsed_ms.count() << " ms\n";
193 }
194
195 else if (argc == 2) { // Если передан файл с матрицей - считываем его
196
197     GetMatrixFromFile( pathToFile: argv[1]);
198
199     if (quantityOfThreads > n || n < 1 || quantityOfThreads < 1) {
200         cout << endl << "Некорректный ввод!" << endl <<
201             "1. Количество потоков должно быть <= размерности матрицы." << endl <<
202             "2. Размерность матрицы должна быть >= 1." << endl <<
203             "3. Количество потоков должно быть >= 1.";
204         return 0;
205     }
206
207     threads = new thread[quantityOfThreads];
208
209     printMatrix(); // вывод матрицы на экран
210     matrixRankFunc(); // функция подсчета ранга матрицы
211
212     cout << endl << "Полученный ранг матрицы: " << rankOfMatrix << endl; // Вывод результатов работы потока для ранга
213
214     auto end = chrono::steady_clock::now();
215     auto elapsed_ms = chrono::duration_cast<std::chrono::milliseconds>(end - begin);
216     cout << "Время работы программы : " << elapsed_ms.count() << " ms\n";
217 }
218
219 return 0;
220 }

```