МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ»

Институт ИЦТМС
Кафедра ИСТАС

КУРСОВОЙ ПРОЕКТ

по дисциплине
«Алгоритмизация. Технология разработки программного обеспечения»

Тема:
«Программное приложение «Приведение разреженной матрицы к ленточной форме – алгоритм Катхилла и Мак-Ки» »

Выполнил студент (институт, курс, группа) ИЦТМСм 1-4 Мунчаев О.М.
(институт (филиал), курс, группа, Ф.И.О.)

Руководитель проекта Доцент, к.т.н., доцент Китайцева Е.Х.
(ученое звание, ученая степень, должность, Ф.И.О.)

К защите _____
(дата, подпись руководителя)

Проект защищен с оценкой

Председатель комиссии _____
(ученое звание, ученая степень, должность, Ф.И.О.)

Члены комиссии: _____
_____
_____
(дата, подпись члена комиссии)

Москва
2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ»

Институт ИЦТМС

Кафедра ИСТАС

Дисциплина Алгоритмизация. Технология разработки программного обеспечения

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА

ФИО обучающегося Мунчаев О.М.
Курс, группа 1-4

1. Тема курсового проекта «Программное приложение «Приведение разреженной матрицы к ленточной форме – алгоритм Катхилла и Мак-Ки»»
2. Сроки сдачи проекта
3. Исходные данные к курсовому проекту: алгоритм Тьюарсона (схема 3) упаковки разреженных матриц_____

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)
   - Описание алгоритма;
   - Структура приложения;
   - Руководство пользователя
5. Перечень графического и иного материала (с точным указанием обязательных чертежей)_____

6. Дата выдачи задания ____30 сентября 2021 г._____

Обучающийся _____
(подпись)

Руководитель _____
(подпись)

Оглавление

ВВЕДЕНИЕ

Разреженная матрица – матрица, имеющая малый процент ненулевых элементов, расположенных случайным образом относительно нулевых элементов. Разреженные матрицы встречаются при решении многих практических задач, а именно: структурного анализа, теории электрических сетей и энергосистем распределения энергии, численного решения дифференциальных уравнений, теории графов, генетики, социологии, программирования для ЭВМ и тд.

С вычислительной точки зрения, работа с ленточными матрицами всегда предпочтительнее работы с аналогичными размерами. квадратные матрицы. Матрицу полос можно сравнить по сложности с прямоугольной матрицей, размер строки которой равен ширине полосы матрицы полосы. Таким образом, объем работы, связанной с выполнением таких операций, как умножение, значительно сокращается, что часто приводит к огромной экономии времени на вычисления и сложность.

Поскольку разреженные матрицы поддаются более эффективным вычислениям, чем плотные матрицы, а также более эффективному использованию компьютерной памяти, было много исследований, направленных на поиск способов минимизировать пропускную способность (или напрямую минимизировать заполнение) путем применения перестановок к матрицу или другие подобные преобразования эквивалентности или подобия.

В Алгоритм Катхилла – Макки может использоваться для уменьшения пропускной способности разреженного симметричная матрица.

Цель курсовой работы заключается в разработке приложения, реализующего алгоритм приведение разреженной матрицы к ленточной форме – алгоритм Катхилла и Мак-Ки.

ОПИСАНИЕ АЛГОРИТМА

Данный алгоритм подразумевает уменьшение ширины ленты соответствующей матрицы, путем изменения номеров вершин.

1. Для каждой вершины i графа Q, соответствующего матрице B, вычислить ее степень p_i, равную общему числу недиагональных единиц 1-й строки матрицы B. Затем выбрать какую-либо вершину i_1 и для которой $p_i=\min_i p_i$, и пометить эту вершину первой.

2. Присвоить вершинам, смежным с вершиной 1, новые номера, начиная с 2, в порядке возрастания их степеней (если степени некоторых смежных вершин совпадают, то выбирать любую из них). Эти вершины относят к первому уровню.

3. Повторить эту процедуру последовательно для каждой из вершин первого уровня — это значит сперва для вершины 2, затем для вершины 3 и т.д.

4. Повторить вышеизложенную процедуру для вершин каждого следующего уровня, пока все n вершин графа Q не будут перенумерованы. Если Q состоит из двух или более несвязных подграфов, то процедура заканчивается, как только все вершины в подграфе перенумерованы. В этом случае необходимо выбрать начальную вершину в каждом из несвязных подграфов и повторить шаги 2, 3, 4 для каждого из них.

5. Наконец, переставить строки и столбцы матрицы B (или A) в соответствии с новыми номерами вершин для получения B (или A).

СТРУКТУРА ПРИЛОЖЕНИЯ

Приложение написано на языке программирования C++20. Исходная матрица хранится в виде графа, представленного списком смежностей. Вершины матрицы выделены в отдельную структуру, для использования ссылок на соседей.

```
40 //структура для вершины
41 struct vertex
42 {
43     int degree; //степень
44     plain_array<vertex_SPtr> neigbords; //массив указателей на соседей
45     plain_array<int> neig_index; // массив индексов соседей
46     unsigned index; //начальный индекс
47     unsigned new_index; // индекс после работы алгоритма
48     plain_array<int> new_neig_index; // новый список соседей
49 
50     vertex(int i, plain_array<vertex_SPtr> n,
51            plain_array<int> l, unsigned inddex,
52            unsigned new_index,plain_array<int> new_neig)
53            :degree(i),neigbords(n),neig_index(l),
54            index(inddex),new_index(new_index),
55            new_neig_index(new_neig) {}
56 };
57 
```

Рис. 1. Структура для вершины.

Сначала создается список смежностей для вершин графа и высчитывается степень каждой вершины.

```
59 //создаю список указателей на вершины из списка смежностей
60 plain_array<vertex_SPtr> create_vertex_array (
61     const plain_array<plain_array<int>> &list_smej,
62     const plain_array<plain_array<int>>& source_matrix)
63 {
64     plain_array<vertex_SPtr> map_vert;
65     for(int i =0;i < list_smej.size(); i++)
66     {
67         int subm = 0;
68         //степень складывается из недиагональных единиц
69         if(source_matrix[i][i] != 0) subm = 1;
70         vertex omar(list_smej[i].size() -1 ,
71                 plain_array<vertex_SPtr>(),list_smej[i],
72                 i,0,plain_array<int>());
73         map_vert.push_back(std::make_shared<vertex>(omar));
74     }
75     for(int i = 0; i< list_smej.size(); i++){
76         map_vert[i]->neigbords = [&map_vert]
77         (plain_array<int> vect_ind) ->plain_array<vertex_SPtr>{
78             plain_array<vertex_SPtr> ret;
79             for(auto &a:vect_ind){ ret.push_back(map_vert[a]);}
80             return ret;}(map_vert[i]->neig_index);
81     }
82     return map_vert;
83 }
84
```

Рис. 2. Функция для создание начального списка смежностей

Также было реализовано несколько вспомогательных функций для удобной работы с графом.

```
85 // достать индекс вершины с минимальной степенью
86 int get_min_degree(const plain_array<vertex_SPtr>& vert_list)
87 {
88     int current = 100000;  //предположу, что так много вершин не будет
89     int index = 0;
90     for(auto& a:vert_list){
91         if(a->degree < current && a->degree != 0){
92         current = a->degree;
93         index = a->index;
94         }
95     }
96     return  index;
97 }
```

Рис. 3. Функция для получения индекса вершины с минимальной степенью

```
//достать вершину по индексу O(n)- долго при большом количестве
vertex_SPtr get_vertex_by_index(
    const int index,
    const plain_array<vertex_SPtr> vert_list){
    for(auto &a:vert_list)
        if(a->index == index)
            return a;
    return nullptr;
}
```

Рис. 4. Функция для получения вершины по индексу

```
//достать не помеченную вершину
vertex_SPtr get_not_mark_vertex(
    const plain_array<vertex_SPtr>& vert_list,
    const std::set<int>& pomech){

    for(const auto&a: vert_list)
        if(!pomech.contains(a->index))
            return a;
    return nullptr;
}
```

Рис. 5. Получение не помеченной вершины

```
//сортировка массива с вершинами по степени, для более удобного
plain_array<vertex_SPtr>
sort_vert_list_by_degree(plain_array<vertex_SPtr> vert_list)
    {
        std::sort(vert_list.begin(),vert_list.end(),
                  [&vert_list]
                  (vertex_SPtr i, vertex_SPtr j){
                      return i->degree < j->degree;
                  });
        return vert_list;
}
```

Рис. 6. Функция для сортировки списка вершин по размеру степени

Реализация алгоритма Катхилл-Макки начинается с поиска вершины с минимальной степенью. Также создается очередь с вершинами, в которые мы будем посещать при проходе по графу и множество помеченных вершин. Начинаем новую нумерацию с нуля.

```
134 //основной алгоритм
135 void cuthill_mckee_algo(plain_array<vertex_SPtr>& vert_list){
136
137     // берем индекс минимальной вершины
138     int current_ind = get_min_degree(vert_list);
139     // кладем ее в набор помеченных вершин(мы там были)
140     std::set<int> pomech ={current_ind};
141     // очередь с вершинами, которые мы должны посетить
142     std::queue<vertex_SPtr> inqueue;
143     // достаем вершину с минимальной степенью
144     auto current_vert = get_vertex_by_index(current_ind,vert_list);
145     if(current_vert == nullptr){
146         std::cerr <<"Bad vertex\n";
147         return ;
148     }
149     // кладем ее в очередь
150     inqueue.push(current_vert);
151     // устанавлимаем ее индекс в 0;
152     current_vert->new_index = 0;
153     int i = 1;
154
```

Рис. 7. Начало основного алгоритма.

Алгоритм Катхилл-Макки является частным случаем обхода в ширину. На каждой итерации мы должны пометить текущую вершину и пройтись по соседям этой вершины. В отличие от стандартного поиска в ширину, необходимо отсортировать список соседей по степеням. Каждому соседу присваивается новый индекс, увеличенный на единицу.

Так как нужно обойти все вершины графа, необходимо избежать проблемы, которая может возникнуть между несвязными частями. Реализация должна учитывать несвязные подграфы основного графа. Проблема решается поиском первой не помеченной вершины, в списке всех вершин.

```
155 ····while(!inqueue.empty()·)
156 ····{
157 |       //достаем·вершину·из·очереди
158 |       current_vert·=·inqueue.front();
159 ········inqueue.pop();
160 |       //·сортируем·список·соседей·в·текущей·вершине
161 ········auto·sort_vertex_list
162 |       ····=·sort_vert_list_by_degree(current_vert->neigbords);
163 ········for(auto&·a:·sort_vertex_list){·//идем·по·всем·соседям
164 |       ····if(pomech.contains(a->index)){
165 |       |       //пропускаем·помеченные
166 |       |       continue;
167 |       ····}else{
168 |       |       //·кладем·в·очередь·не·помеченные
169 |       |       inqueue.push(a);
170 |       |       pomech.insert(a->index);
171 |       |       //·увеличиваем·текущий·индекс
172 |       |       a->new_index·=·i++;
173 |       ····}
174 |       }
175 |       //·проверка·при··несвязных·подграфах·в·основном·графе
176 |       if(inqueue.empty()){
177 |       ····//·достаем·первую·попавшуюся·не·помеченную·вершину
178 |       ····auto·temp·=·get_not_mark_vertex(vert_list,·pomech);
179 |       ····if(temp){
180 |       |       pomech.insert(temp->index);
181 |       |       temp->new_index·=·i++;
182 |       |       inqueue.push(temp);
183 |       ····}
184 |       }
185 ····}
```

Рис. 8. Реализация обхода в ширину в основном алгоритме.

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Технические требования для сборки и запуска приложения:

1) Компьютер

2) ОС - Gnu-Linux/MacOS.

3) Компилятор - GCC/Clang с поддержкой C++20(concepts)

4) Командная оболочка(Bash)

Для сборки проекта был написан Makefile.

```
all:
	clang++-13 -g  -std=c++20 -fno-rtti  -I./ -o final_alg final_alg.cpp

```

Рис. 9. Makefile для сборки проекта

Чтобы собрать проект необходимо воспользоваться стандартной Gnu утилитой make. Введите make в вашей командоной оболочке, после этого создаться бинарный файл final_alg, который и является программой для работы запуска алгоритма.

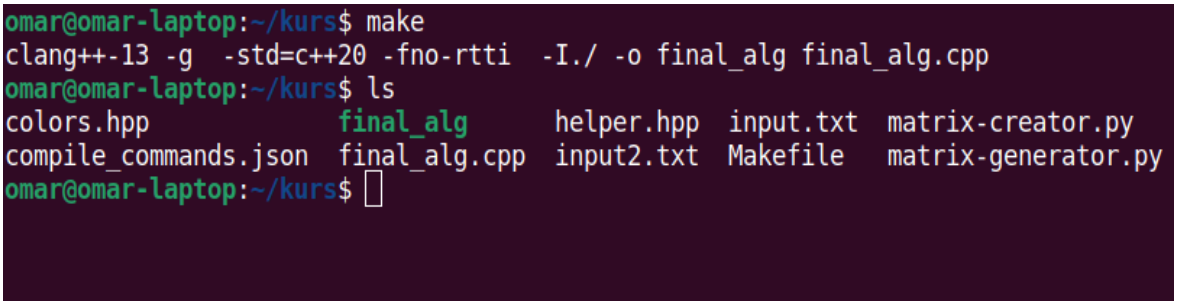

Рис. 10. Запуск сборки проекта и получение файла final_alg.

Исходная матрица 289x289 была извлечена из Excel файла и приведена к обычному текстовому файлу с нулями и единицами. При выводе, единицы подсвечиваются более темным цветом(синим).

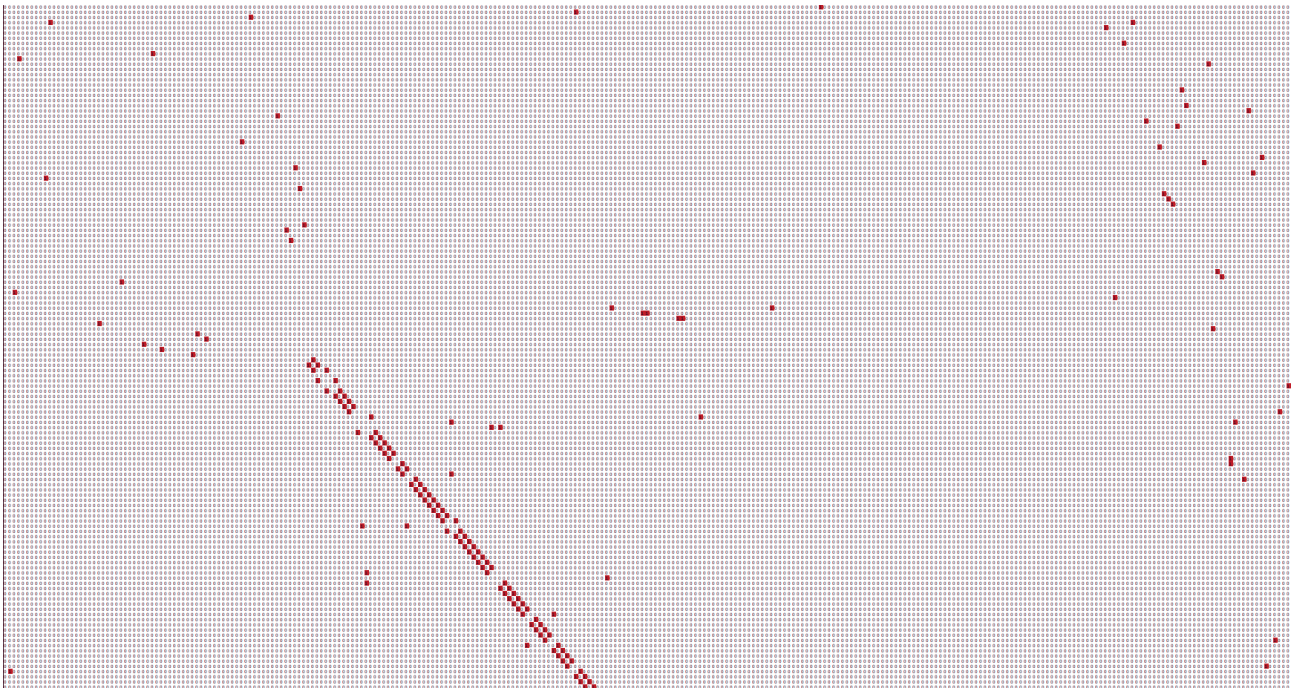

Рис. 11. Первая часть исходной матрицы

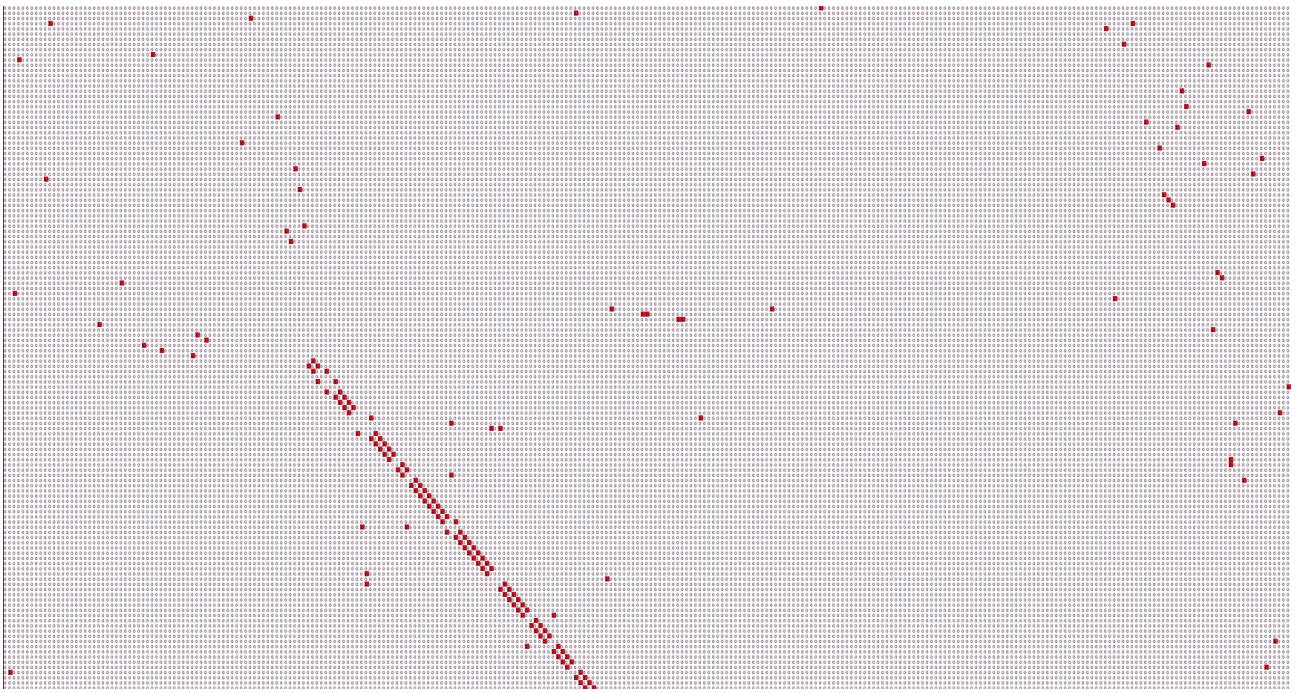

Рис. 12. Вторая часть исходной матрицы

Для запуска приложения, необходимо подать на вход текстовый файл с матрицей. Введите команду ./final_alg и следующим аргументом укажите файл с исходными данными.

```
omar@omar-laptop:~/kurs$ ./final_alg source_input.txt
289 289
```

Рис. 13. Запуск программного приложения

В результате мы получим матрицу, приведеную к ленточной форме.

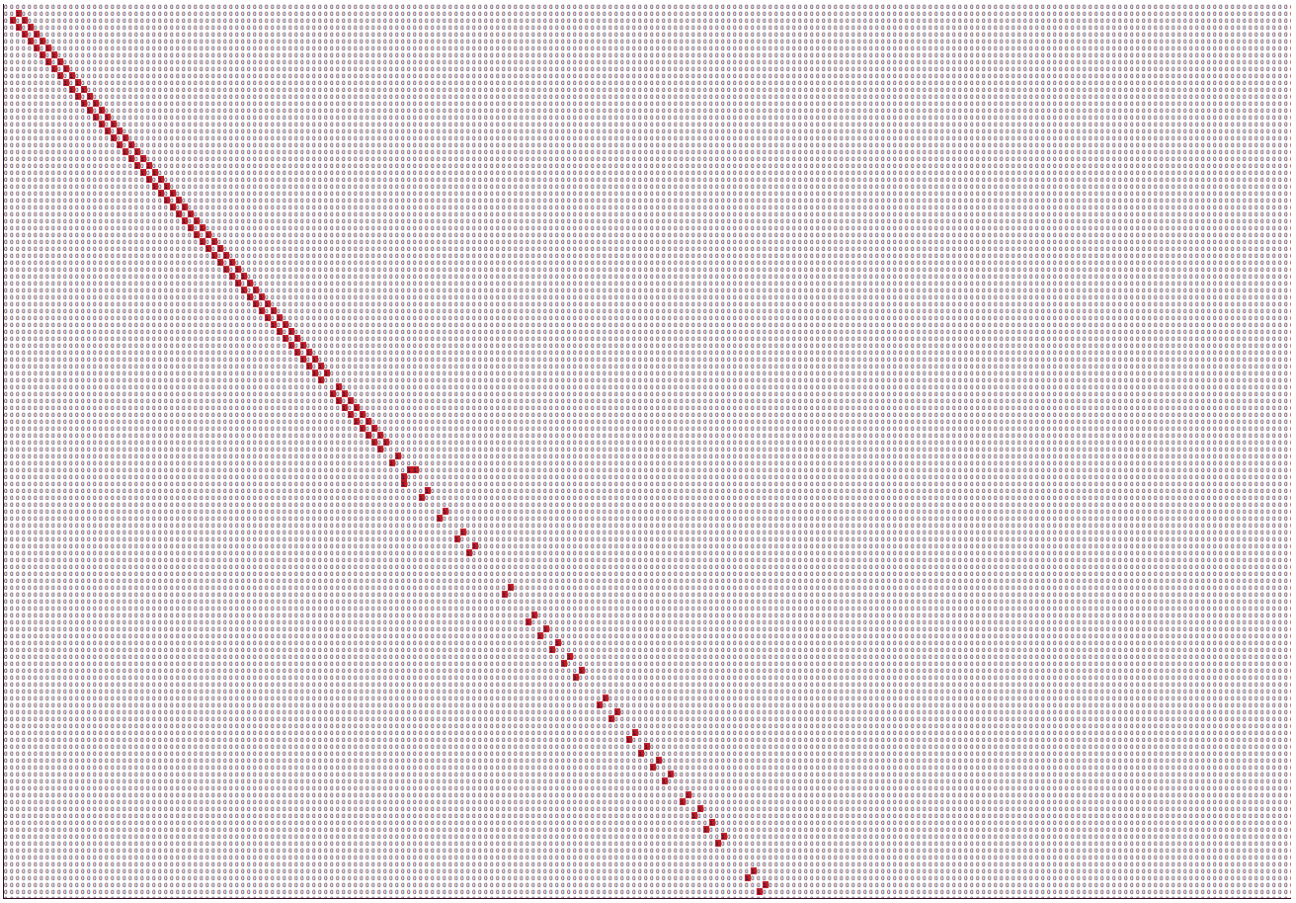

Рис. 14. Первая часть матрицы, полученной после работы программы

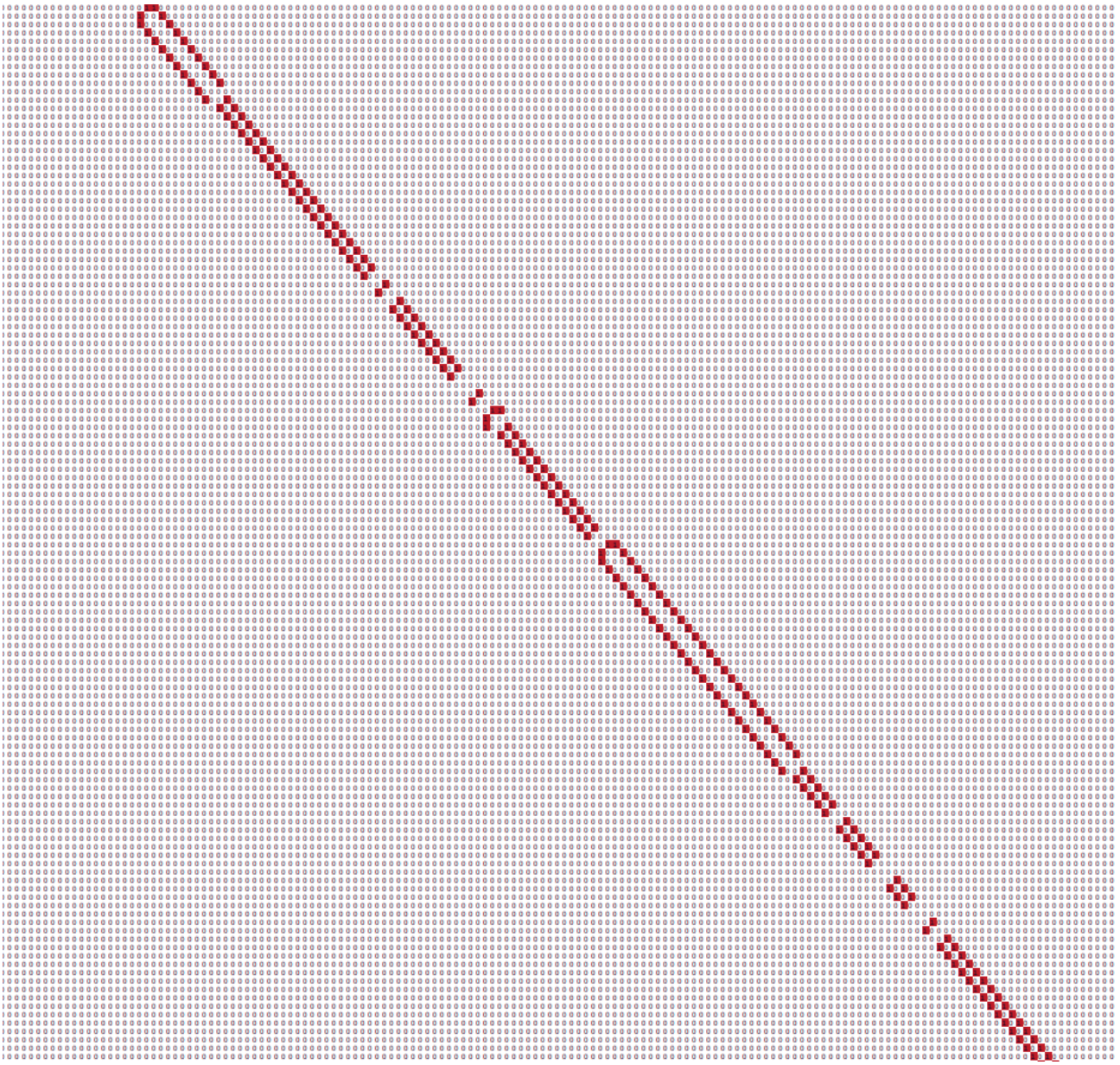

Рис. 15. Вторая часть матрицы, полученной после работы программы

Добавим в исходную матрицу несколько элементов так, чтобы они расположились на одной строке(столбце).

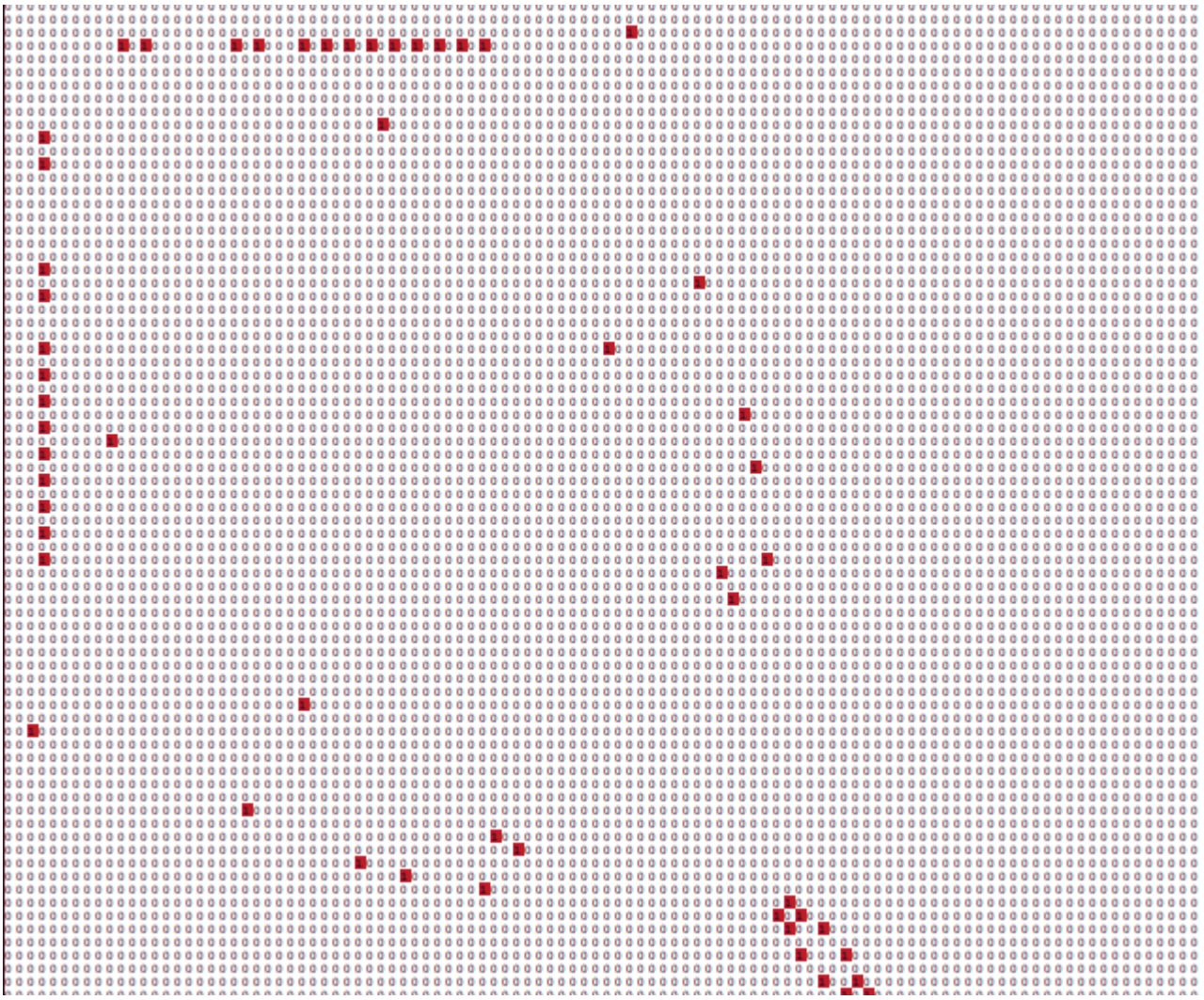

Рис. 16. Добавление новых соседей к одной вершине графа

После такого преобразования, граф приобретет вид, изображенный на рисунке 17.

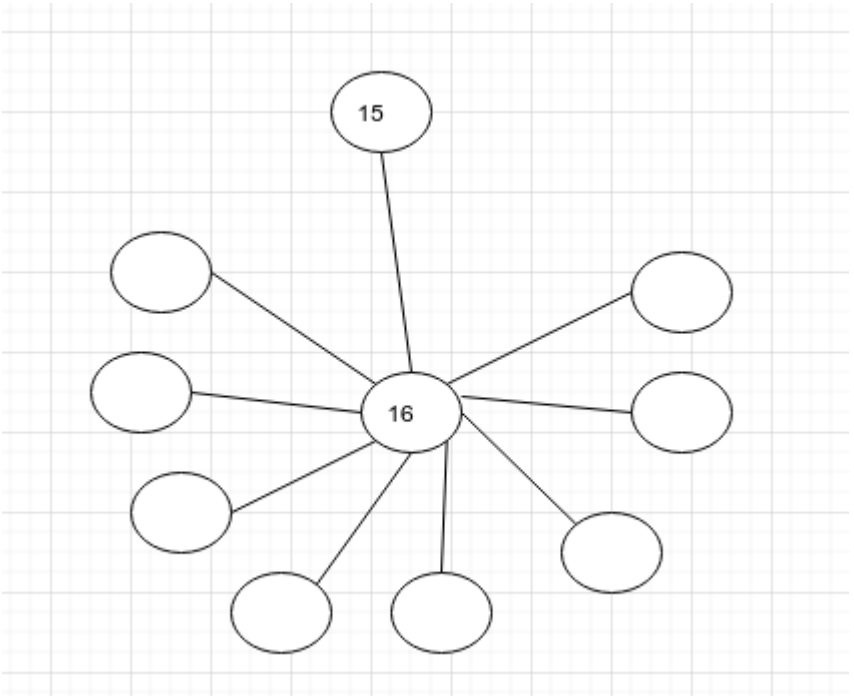

Рис. 17 Пример подграфа в искомом графе

Алгоритм Катхилла-Макки плохо справляется с такими участками, и ширина ленты становится равна числу смежных вершин для вершины 16.

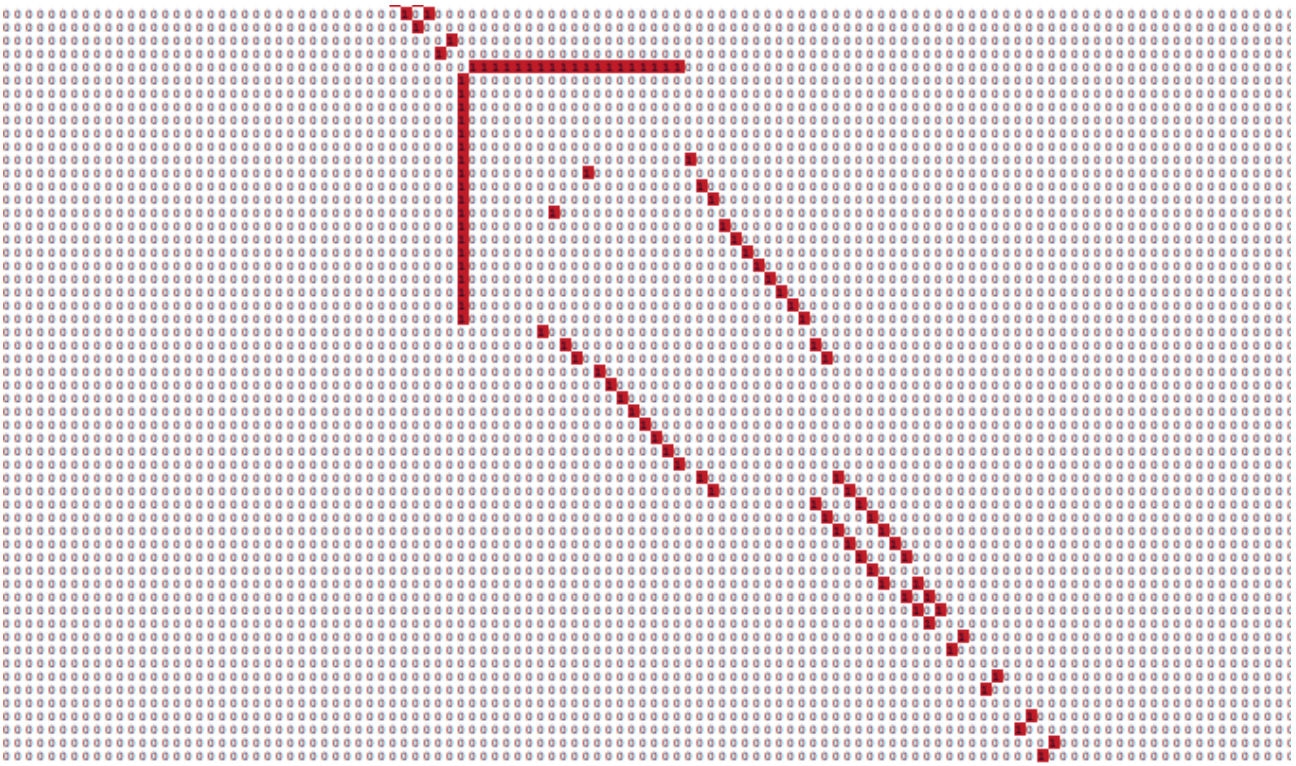

Рис. 18. Участок ленты, для вершины, с большим количеством соседей

```
Ширина ленты искомой матрицы: 285
Ширина ленты преобразованной матрицы:19
omar@omar-laptop:~/kurs$
```

Рис. 19. Результат работы программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано приложения на языке программирования C++, реализующее уменьшение ширины матрицы, с помощью алгоритма Катхилл-Макки. Исходный код проекта расположен по ссылке: https://github.com/bigmuncha/cuthill-mckee-algo

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Тьюарсон Р. Разреженные матрицы – М.: Мир, 1977. – 191 с.
2. Писсанецки С. Технология разреженных матриц – М.: Мир, 1988. – 410 с.
3. Стивенс Алгоритмы. Теория и практическое применение М.: Издательство “Э”, 2016.-544 с.
4. Самоучитель C++ (21 серия) Visual Studio, Матрицы и многомерные массивы [Электронный ресурс] URL: https://www.youtube.com/watch?v=2op17CTY0Bc&ab_channel=ProgTeachTV (Дата обращения: 15.12.2021)
5. Садовский Б.С. Технология программирования, 2009. – 98 с.
6. Довек, Ж. Введение в теорию языков программирования / Ж. Довек, Ж.-Ж. Леви. — М.: ДМК, 2016. — 134 с.
7. Довбуш, Галина Visual C++ на примерах / Галина Довбуш , Анатолий Хомоненко. - М.: БХВ-Петербург, 2012. - 528 с
8. Боровский, А.Н. Qt4.7+. Практическое программирование на C++. / А.Н. Боровский. - СПб.: BHV, 2012. - 496 с.
9. Хенкеманс, Д. Программирование на C++ / Д. Хенкеманс, М. Ли. - СПб.: Символ-плюс, 2015. - 416 с.
10. Алистер Коберн - Современные методы описания функциональных требований к системам: Издательство “Лори”, 2012. - 264 с.

ПРИЛОЖЕНИЕ А

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include "helper.hpp"
#include <memory>
#include <algorithm>
#include <ranges>
#include <assert.h>



//создаю список смежностей из матрицы
inline std::vector<std::vector<int>>
create_list_smej(const std::vector<std::vector<int>>& matr)
{
   std::vector<std::vector<int>> res;
   for(int i=0; i < matr.size(); i++)
   {
        std::vector<int> temp;
        for(int j =0; j < matr[i].size(); j++)
        {
           if(matr[i][j] == 1){
                temp.push_back(j);
           }
        }
```

```
        res.push_back(temp);
    }
    return res;
}



class vertex;
//typedef для удобной работы
typedef std::shared_ptr<vertex> vertex_SPtr; //указатель на вершину

template<class T>
using plain_array = std::vector<T>;

//структура для вершины
struct vertex
{
    int degree; //степень
    plain_array<vertex_SPtr> neigbords; //массив указателей на соседей
    plain_array<int> neig_index; // массив индексов соседей
    unsigned index; //начальный индекс
    unsigned new_index; // индекс после работы алгоритма
    plain_array<int> new_neig_index; // новый список соседей

        vertex(int i, plain_array<vertex_SPtr> n, plain_array<int> l, unsigned
inddex,unsigned new_index,plain_array<int> new_neig)
```

```
:degree(i),neigbords(n),neig_index(l),index(inddex),new_index(new_index),new_neig_index(new_neig) {}
};


//создаю список указателей на вершины из списка смежностей
plain_array<vertex_SPtr> create_vertex_array (const plain_array<plain_array<int>> &list_smej,const plain_array<plain_array<int>>& source_matrix)
{
   plain_array<vertex_SPtr> map_vert;
   for(int i =0;i < list_smej.size(); i++)
   {
       int subm = 0;
       if(source_matrix[i][i] != 0) subm = 1; //степень складывается из недиагональных единиц
       vertex omar(list_smej[i].size() -1 ,plain_array<vertex_SPtr>(),list_smej[i],i,0,plain_array<int>());
       map_vert.push_back(std::make_shared<vertex>(omar));
   }
   for(int i = 0; i< list_smej.size(); i++){
       map_vert[i]->neigbords = [&map_vert](plain_array<int> vect_ind) ->plain_array<vertex_SPtr>{
               plain_array<vertex_SPtr> ret;
               for(auto &a:vect_ind){ ret.push_back(map_vert[a]);}
               return ret;}(map_vert[i]->neig_index);
   }
   return map_vert;
```

```
}

// достать индекс вершины с минимальной степенью
int get_min_degree(const plain_array<vertex_SPtr>& vert_list)
{
   int current = 100000;  //предположу, что так много вершин не будет
   int index = 0;
   for(auto& a:vert_list){
        if(a->degree < current && a->degree != 0){
           current = a->degree;
           index = a->index;
        }
   }
   return  index;
}

// достать вершину по индексу O(n)- долго при большом количестве вершин, можно использовать другой контейнер
vertex_SPtr  get_vertex_by_index(const  int  index,const  plain_array<vertex_SPtr> vert_list){
   for(auto &a:vert_list)
        if(a->index == index)
           return a;
   return nullptr;
}

// достать не помеченную вершину
```

```
vertex_SPtr get_not_mark_vertex(const plain_array<vertex_SPtr>& vert_list,const std::set<int>& pomech){

    for(const auto&a: vert_list)
        if(!pomech.contains(a->index))
            return a;
    return nullptr;
}


//сортировка массива с вершинами по степени, для более удобного перебора
plain_array<vertex_SPtr> sort_vert_list_by_degree(plain_array<vertex_SPtr> vert_list){
        std::sort(vert_list.begin(),vert_list.end(),
                [&vert_list]
                (vertex_SPtr i, vertex_SPtr j){
                    return i->degree < j->degree;
                });
        return  vert_list;
}


//основной алгоритм
void cuthill_mckee_algo(plain_array<vertex_SPtr>& vert_list){

      int current_ind = get_min_degree(vert_list); // берем индекс минимальной вершины
    std::cout <<"in cuthill mccke"<< current_ind <<"\n";
```

```
        std::set<int> pomech ={current_ind}; // кладем ее в набор помеченных вершин(мы там были)
    std::queue<vertex_SPtr> inqueue; // очередь с вершинами, которые мы должны посетить
        auto current_vert = get_vertex_by_index(current_ind,vert_list); // достаем вершину с минимальной степенью
   if(current_vert == nullptr){
        std::cerr <<"Bad vertex\n";
        return ;
   }
   inqueue.push(current_vert); // кладем ее в очередь
   current_vert->new_index = 0; // устанавлимаем ее индекс в 0;
   int i = 1;
   std::cout <<"here\n";
   std::cout <<pomech.size() <<" " << vert_list.size() << inqueue.empty();
   while(!inqueue.empty() )
   {
        current_vert = inqueue.front(); //достаем вершину из очереди
        std::cout <<"current vertex" << current_vert->index<< " neigbords: ";
        inqueue.pop();
        for(auto&a:current_vert->neigbords){
           std::cout << a->index <<' ';
        }
        auto sort_vertex_list = sort_vert_list_by_degree(current_vert->neigbords); // сортируем список соседей в текущей вершине
        std::cout << "sort: ";
        for(auto&a:sort_vertex_list){
           std::cout << a->index <<' ';
```

```
        }
        std::cout <<'\n';
        for(auto& a: sort_vertex_list){ //идем по всем соседям
           if(pomech.contains(a->index)){
                continue; //пропускаем помеченные
           }else{
                inqueue.push(a); // кладем в очередь не помеченные
                pomech.insert(a->index);
                a->new_index = i++; // увеличиваем текущий индекс
           }
        }
        // проверка при  несвязных подграфах в основном графе
        if(inqueue.empty()){
             auto temp = get_not_mark_vertex(vert_list, pomech); // достаем первую попавшуюся не помеченную вершину
           if(temp){
                pomech.insert(temp->index);
                temp->new_index = i++;
                inqueue.push(temp);
           }
        }
   }
   std::cout<<"vert_list_size "<<vert_list.size() - pomech.size() <<" "<< i <<"\n";
   for(int i=0; i < vert_list.size(); i++){
        plain_array<int> new_neig; // создаем для каждой вершины новый список соседей
        for(auto &a:vert_list[i]->neig_index){
           auto cur = get_vertex_by_index(a, vert_list);
```

```
            if(cur)
                vert_list[i]->new_neig_index.push_back(cur->new_index);
            else
                std::cerr <<" error in get vert\n";
        }
    }
}

// создаем матрицу из списка смежностей вершин
plain_array<plain_array<int>> get_matrix(const plain_array<vertex_SPtr>&
vert_list)
{
                                        plain_array<plain_array<int>          >
ret_val(vert_list.size(),plain_array<int>(vert_list.size(),0));
    for(auto&a:vert_list){
        for(auto&s:a->new_neig_index){
            ret_val[a->new_index][s] = 1;
        }
    }
    return ret_val;
}

void main_algo(plain_array<plain_array<int>> matrix)
{
     prints(matrix); //вывод изначальной матрицы
    auto list_smej = create_list_smej(matrix); //создаю из нее список смежностей
      auto vertex_array = create_vertex_array(list_smej,matrix); // создаю из списка
смежностей массив с вершинами
```

```
    cuthill_mckee_algo(vertex_array);
    auto new_matr = get_matrix(vertex_array);
    prints(new_matr); // вывожу результат
}



int main(int argc, char **argv)
{
    if(argc != 2){
        std::cerr<<" bad argc \n" << argc;
        return 0;
    }
    auto matr = fileIn<int>(argv[1]);
    std::cout <<matr.size() <<" " << matr[0].size() <<'\n';
    main_algo(matr);
    return 0;
}
```