**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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

* Описание алгоритма;
* Структура приложения;
* Руководство пользователя

1. Перечень графического и иного материала (с точным указанием обязательных чертежей)\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Дата выдачи задания \_\_\_\_30 сентября 2021 г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**Оглавление**

[ВВЕДЕНИЕ 4](#__RefHeading___Toc2449_512189521)

[ОПИСАНИЕ АЛГОРИТМА 5](#__RefHeading___Toc2451_512189521)

[СТРУКТУРА ПРИЛОЖЕНИЯ 6](#__RefHeading___Toc2453_512189521)

[РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 11](#__RefHeading___Toc2455_512189521)

[ЗАКЛЮЧЕНИЕ 15](#__RefHeading___Toc2457_512189521)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 16](#__RefHeading___Toc2459_512189521)

[ПРИЛОЖЕНИЕ А 17](#__RefHeading___Toc2461_512189521)

# **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 В, вычислить ее степень pi, равную общему числу недиагональных единиц 1-й строки матрицы В. Затем выбрать какую-либо вершину i1 и для которой pi=mini pi, 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 верщин 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 В (или А) в соответствии с новыми номерами вершин для получения В (или А).

# **СТРУКТУРА ПРИЛОЖЕНИЯ**

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

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

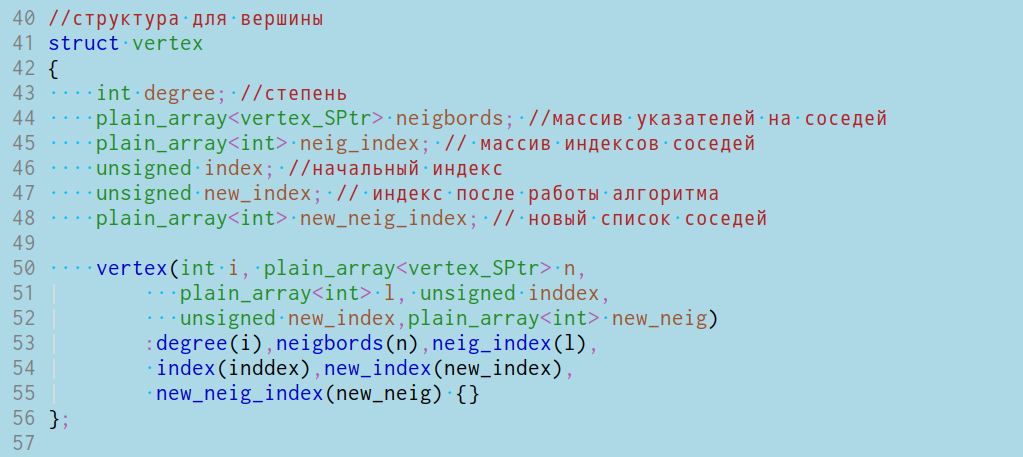
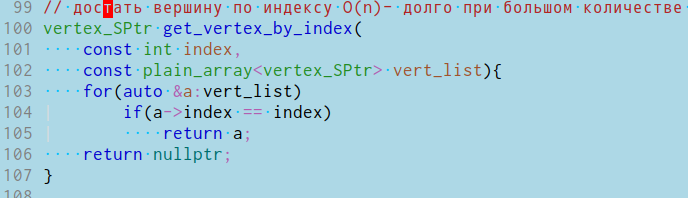
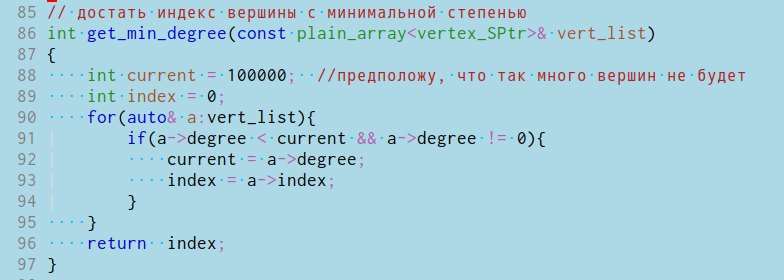
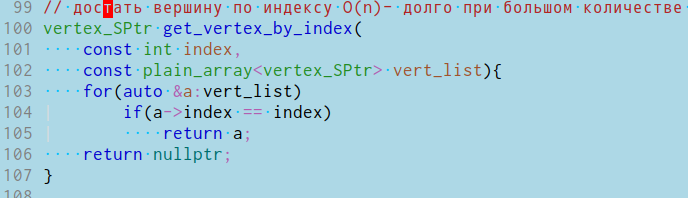
Рис. 1. Структура для вершины.

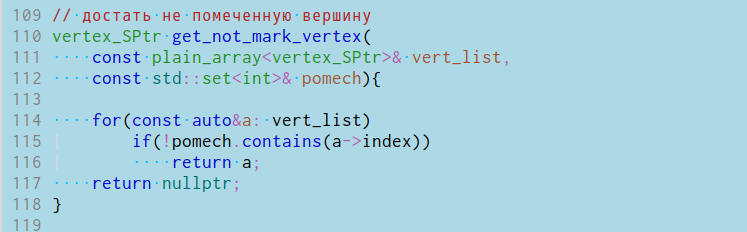
Рис. 2. Функция для создание начального списка смежностей

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

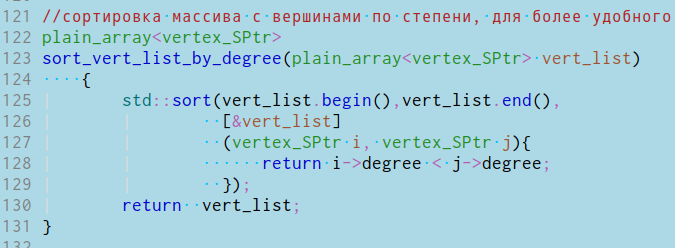


Рис. 3. Функция для получения индекса вершины с минимальной степенью

Рис. 4. Функция для получения вершины по индексу

Рис. 5. Получение непомеченной вершины

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.

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

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.

Рис. 7. Начало основного алгоритма.

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.

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

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Технические требования для сборки и запуска приложения:

1) Компьютер

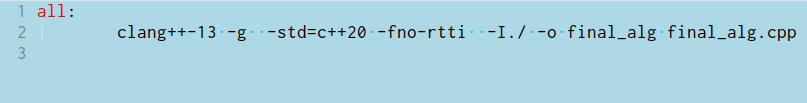
2) ОС - Gnu-Linux/MacOS.

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

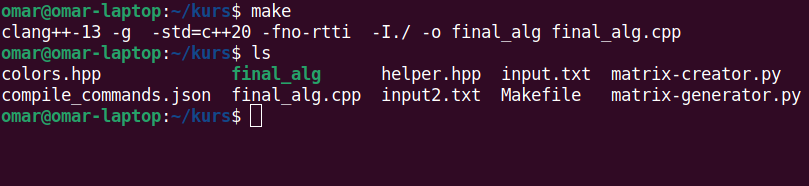
4) Командная оболочка(Bash)

Для сборки проекта был написан Makefile.

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. 9. Makefile для сборки проекта

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. 10. Запуск сборки проекта и получение файла final\_alg.

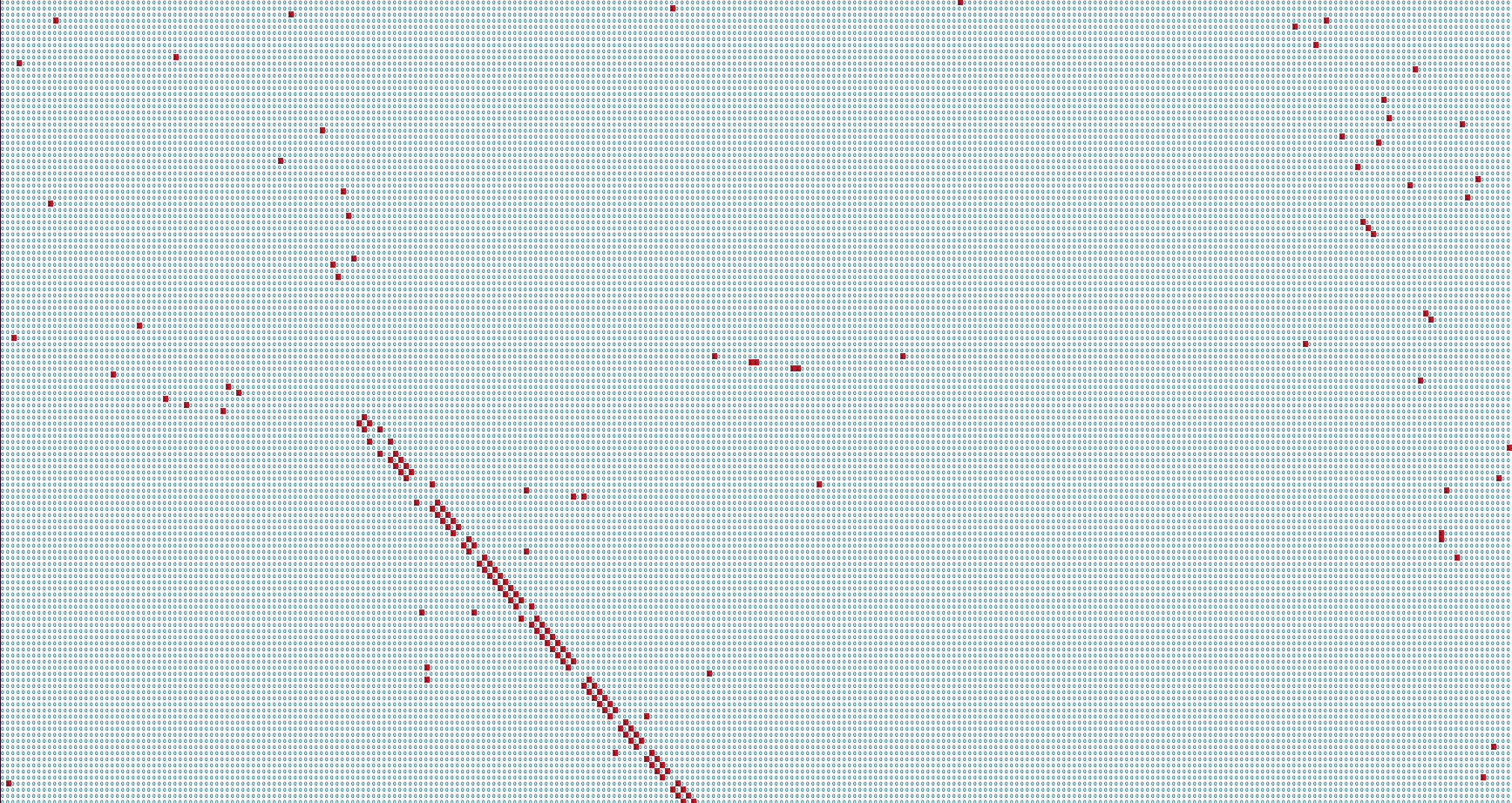


Рис. 11. Первая часть исходной матрицы

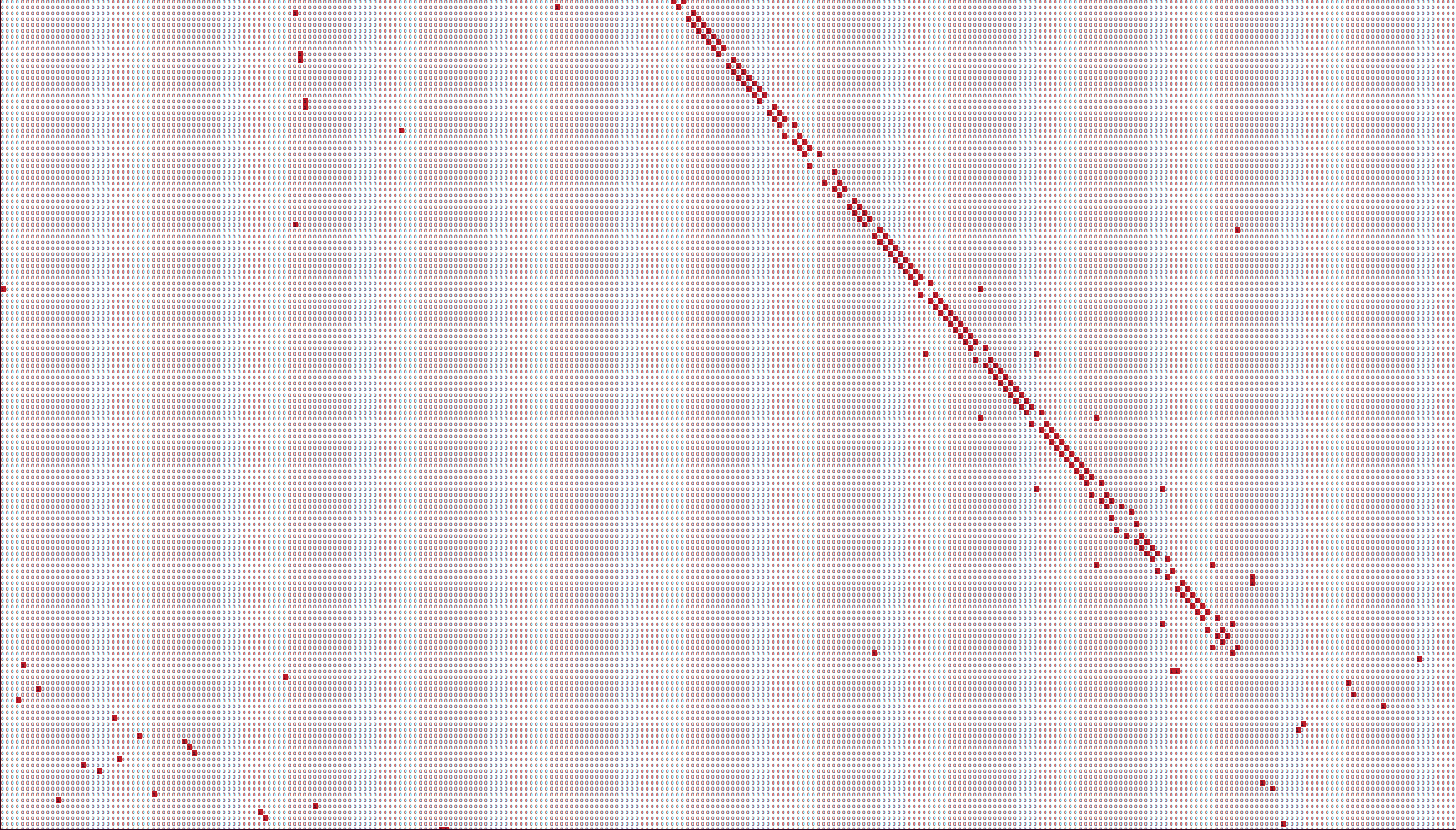


Рис. 12. Вторая часть исходной матрицы.

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

Рис. 13. Запуск программного приложения

В результате мы получим матрицу, приведеную к ленточной форме.

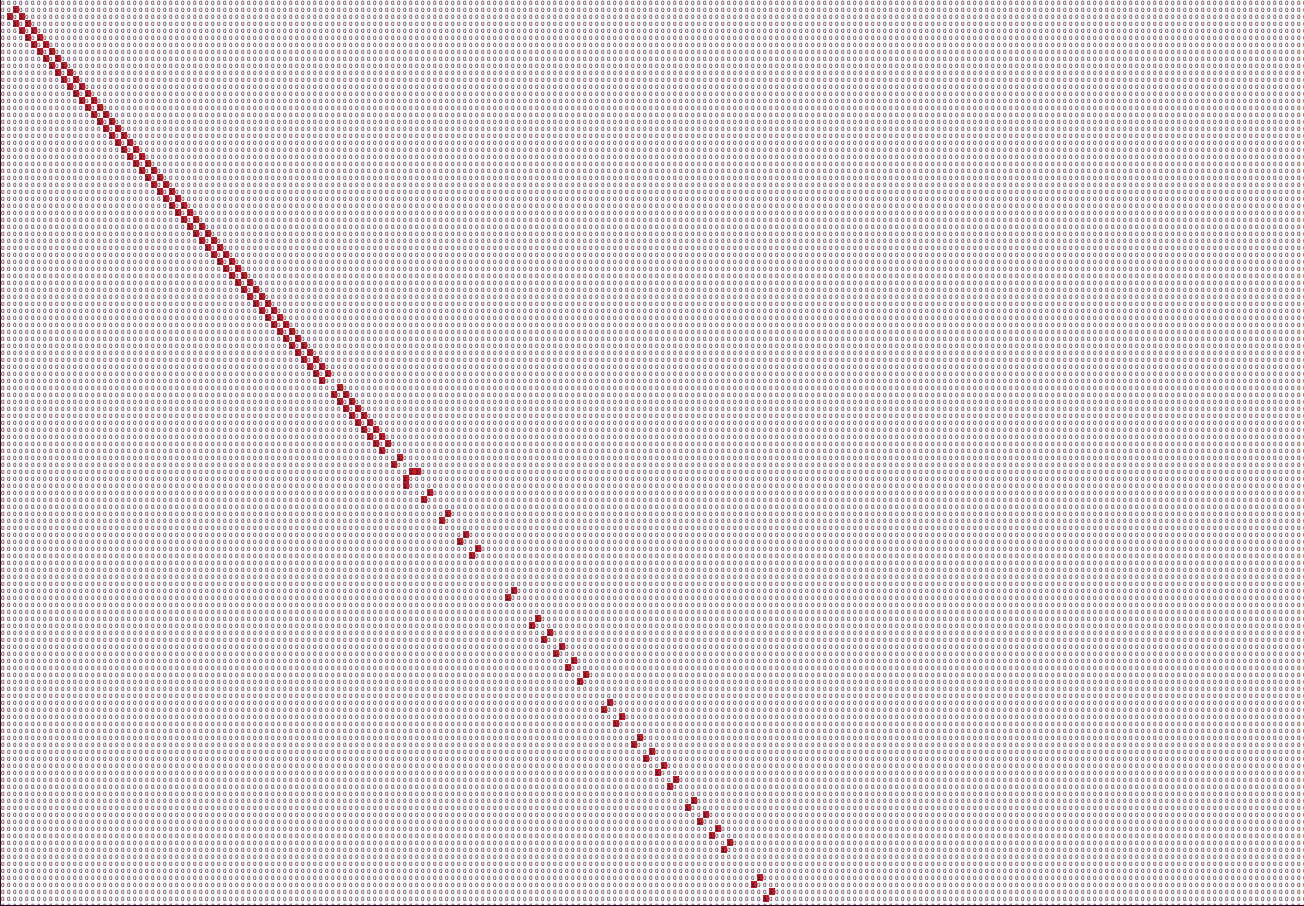


Рис. 14. Первая часть матрицы, полученной после работы программы

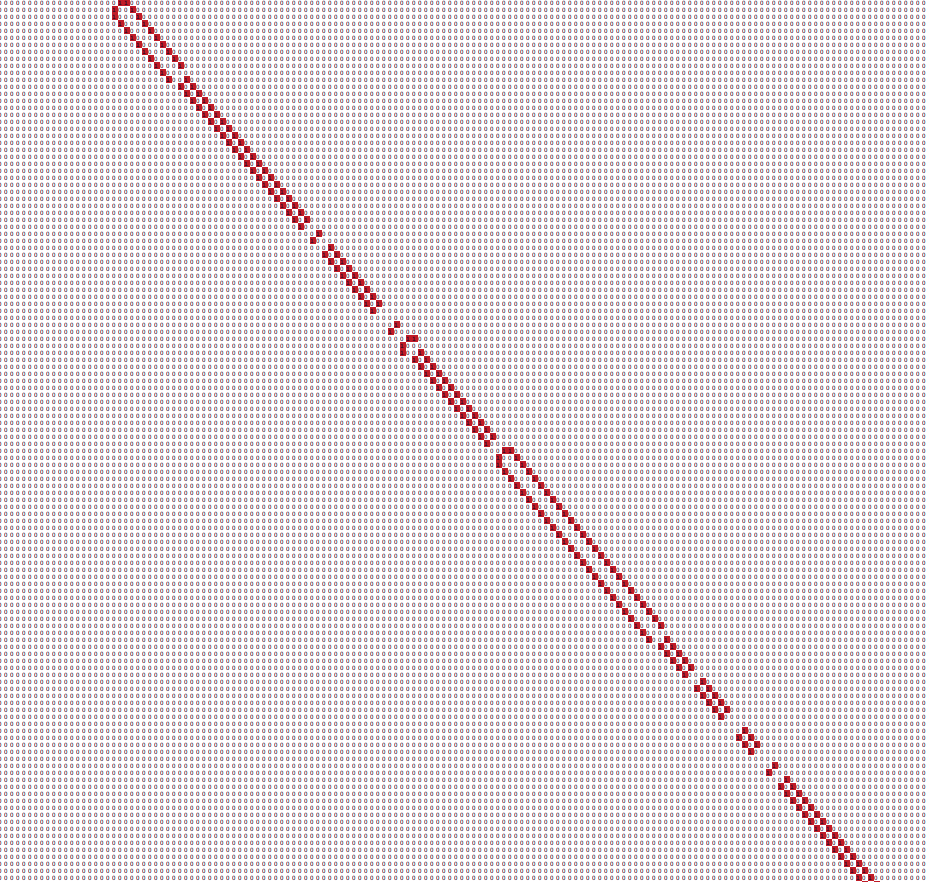
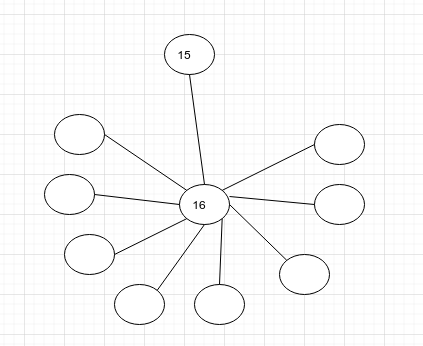


Рис. 15. Вторая часть матрицы, полученной после работы программы

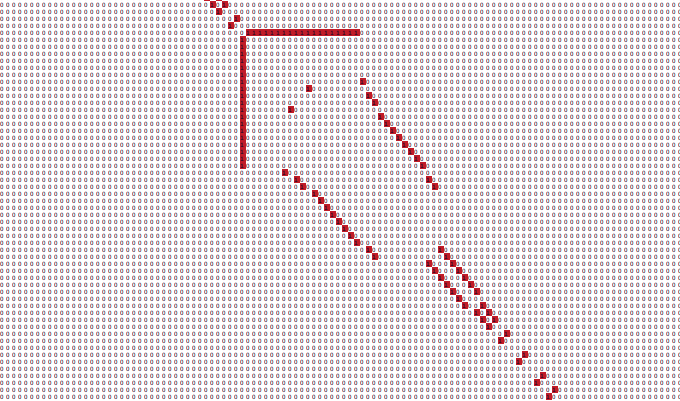
# Добавим в исходную матрицу несколько элементов так, чтобы они расположились на одной строке(столбце).

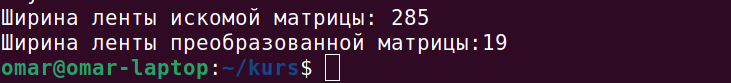
После такого преобразования, граф приобритет вид, изображенный на рисунке 17.

Рис. 16. Добавление новых соседей к одной вершине графа

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

Рис. 19. Результат работы программы

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы было разработано приложения на языке программирования С++, 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 c.
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 c.
7. Довбуш, Галина Visual C++ на примерах / Галина Довбуш , Анатолий Хомоненко. - М.: БХВ-Петербург, 2012. - 528 c
8. Боровский, А.Н. Qt4.7+. Практическое программирование на C++. / А.Н. Боровский. - СПб.: BHV, 2012. - 496 c.
9. Хенкеманс, Д. Программирование на C++ / Д. Хенкеманс, М. Ли. - СПб.: Символ-плюс, 2015. - 416 c.
10. Алистер Коберн - Современные методы описания функциональных требований к системам: Издательство “Лори”, 2012. - 264 c.

## 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;

}