|  |  |  |
| --- | --- | --- |
| Федеральное государственное автономное образовательное учреждение  высшего образования «Национальный исследовательский  Нижегородский государственный университет им. Н.И. Лобачевского» НИИМ Нижегородского университета | | |
| УДК  № госрегистрации  Инв. № | | **УТВЕРЖДАЮ**  Зав. кафедрой ИАНИ ИИТММ  ННГУ д.т.н., профессор  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Х. Прилуцкий  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_ 2017 |
| Научно-технический отчет  **РАЗРАБОТКА ПО ДЛЯ РЕШЕНИЯ ЗАДАЧИ**  **ВОССТАНОВЛЕНИЯ НУМЕРАЦИИ РЕГУЛЯРНОЙ СЕТКИ** | | |
|  |  | |
|  | Доцент каф. ИАНИ ИИТММ ННГУ,  д.т.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Н.В. Старостин \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. | |
|  |  | |
|  |  | |

2017

**СПИСОК ИСПОЛНИТЕЛЕЙ:**

Магистранты 2-го года обучения по направлению «Прикладная информатика»:

1. Алявдин Александр
2. Ильин Сергей
3. Каримов Динар
4. Любимцев Дмитрий
5. Полунин Дмитрий
6. Шестова Александра
7. Шуланкина Елизавета

РЕФЕРАТ

Отчёт 36 с., 37 рис., 2 табл.

Ключевые слова: регулярная сетка, граф, восстановление геометрической информации.

Рассматривается проблема ….. Целью работы является …..

В рамках данного проекта проведены следующие работы….

**СОДЕРЖАНИЕ**

[Введение 9](#_Toc501461735)

[1. Математическая постановка задач распознавания и нумерации 11](#_Toc501461736)

[1.1. Исходные данные. 11](#_Toc501461737)

[1.2. Размерность в задаче распознавания и нумерации 11](#_Toc501461738)

[1.3. Математическая модель 11](#_Toc501461739)

[1.3.1. Одномерный случай 12](#_Toc501461740)

[1.3.2. Двумерный случай 12](#_Toc501461741)

[1.3.3. Трехмерный случай 13](#_Toc501461742)

[1.4. Критерии задачи нумерации 14](#_Toc501461743)

[2. Алгоритмы распознавания и нумерации 15](#_Toc501461744)

[2.1. Необходимые условия распознавания 15](#_Toc501461745)

[2.2. Концепция схемы решения задачи нумерации 15](#_Toc501461746)

[2.3. Описание алгоритма для одномерного случая 16](#_Toc501461747)

[2.4. Описание алгоритма для двумерного случая 16](#_Toc501461748)

[2.5. Описание алгоритма для трехмерного случая 17](#_Toc501461749)

[2.6. Вычислительная сложность алгоритмов распознавания и нумерации 18](#_Toc501461750)

[3. Программная реализация 19](#_Toc501461751)

[3.1. Требования к программе распознавания и нумерации сеток 19](#_Toc501461752)

[3.2. Требования к исходным данным 19](#_Toc501461753)

[3.3. Требования к форматам представления результата 19](#_Toc501461754)

[3.4. Структура программы 20](#_Toc501461755)

[3.5. Описание программы 21](#_Toc501461756)

[4. Тестовый базис и верификация программы 22](#_Toc501461757)

[4.1. Тестовые графы – одномерные сетки 22](#_Toc501461758)

[4.2. Тестовые графы – двумерные сетки 23](#_Toc501461759)

[4.3. Тестовые графы – трехмерные сетки 29](#_Toc501461760)

[4.4. Тестовые графы – не сетки 31](#_Toc501461761)

[4.5. Тестовые графы для тестов производительности 35](#_Toc501461762)

[5. Верификация и тестирование программы 36](#_Toc501461763)

[5.1. Методика верификации 36](#_Toc501461764)

[5.2. Результаты верификации 36](#_Toc501461765)

[Табл 5.1. Результаты верификации 40](#_Toc501461766)

[5.3. Методика проведения тестов производительности 40](#_Toc501461767)

[5.4. Результаты тестов производительности 41](#_Toc501461768)

[Табл 5.1. Результаты тестов производительности 42](#_Toc501461769)

[5.5. Выводы по результатам верификации и тестирования программы 42](#_Toc501461770)

[Заключение. 43](#_Toc501461771)

**ОПРЕДЕЛЕНИЯ**

|  |  |
| --- | --- |
| Виртуальный блок | часть грани, для которой создание регулярной сетки тривиально |
| Срединная поверхность | Геометрический объект – поверхность которая, описывается геометрическим местом точек центров всевозможных шаров, вписанных в твердое тело. |
| ЛОГОС-ПП | программный модуль пакета программ "ЛОГОС" для автоматизированной подготовки и обработки расчётных моделей |

# Введение

Во многих областях используются расчетные аппроксимационные сетки при численном физико-математическом моделировании физических процессов и систем.

При работе с большеразмерными сеточными структурами возникает необходимость в их декомпозиции. В связи с этим имеют место процессы, в рамках которых происходит потеря геометрической информации вершин сетки.

В данном случае рассматривается регулярная сетка.

Регулярная сетка заданной размерности k состоит из n узлов. Каждому узлу сетки присвоим номер, состоящий из k компонент (). Каждый узел связан с соседними узлами. Под регулярностью будем понимать:

1. Все компоненты номеров соседних узлов одинаковые с точностью до одной компоненты, при этом значения этих компонент отличаются на единицу (такие компоненты будем называть близкими);
2. Для любой пары узлов с близкими компонентами соответствующие им вершины являются смежными.

Цель проекта – создание программно-алгоритмических решений, которые обеспечивают решение задачи о восстановлении геометрической информации для вершин графа в рамках регулярной сетки.

Задачи:

1. Формальная постановка задачи и исследование;
2. Создание тестовой инфраструктуры;
3. Написание библиотеки, которая позволяет:
   1. Проверить граф на соответствие необходимым условиям регулярности;
   2. Восстановить нумерацию в случае регулярности графа.
4. Написание консольного приложения, которое позволяет:
   1. Считать исходный граф, представленный в METIS формате;
   2. Проверить граф на регулярность и восстановить нумерацию;
   3. Сохранить результат в JSON формате.

# 1. Математическая постановка задач распознавания и нумерации

## 1.1. Исходные данные

Дан неориентированный помеченный граф G = (V, E), где V - множество вершин и E - множество ребер. Матрица смежности для графа с n вершинами – это квадратная матрица A порядка n:

(1.1.1)

(1.1.2)

(1.1.3)

Матрица смежности симметрична и на главной диагонали нули:

(1.1.4)

(1.1.5)

## 1.2. Размерность в задаче распознавания и нумерации

Размерностью в задаче распознавания и нумерации графа является количество компонент в номере узла сетки.

k – размерность соответствующей регулярной сетки:

(1.2.1)

## 1.3. Математическая модель

Решением является матрица Х размерности , где – значение j-ой компоненты индекса регулярной сетки для i-ой вершины.

(1.3.1)

(1.3.2)

Граф G должен быть связный, все компоненты номеров соседних узлов сетки должны быть одинаковые с точностью до одной компоненты, при этом значения этих компонент отличаются на единицу (такие номера будем называть близкими). Для любой пары узлов с близкими номерами соответствующие им вершины являются смежными.

(1.3.3)

### 1.3.1. Одномерный случай

Граф является линейным. Каждый номер состоит из одной компоненты:

(1.3.1.1)

Пример:

Граф состоит из 3 вершин. Каждой вершине присвоен индекс регулярной сетки.



Рис. 1.1. Пример одномерного случая

### 1.3.2. Двумерный случай

Каждый номер состоит из двух компонент:

Пример:

Граф состоит из 4 вершин. Каждой вершине присвоен индекс регулярной сетки.



Рис. 1.2. Пример двумерного случая

### 1.3.3. Трехмерный случай

Каждый номер состоит из трех компонент:

Пример:

Граф состоит из 7 вершин. Каждой вершине присвоен индекс регулярной сетки.



Рис. 1.3. Пример трехмерного случая

## 1.4. Критерии задачи нумерации

Необходимо пронумеровать граф индексами наименьшей возможной размерности:

(1.4.1)

Количество отсутствующих связей между близкими узлами должно быть минимально:

(1.4.2)

# 2. Алгоритмы распознавания и нумерации

## 2.1. Необходимые условия распознавания

Необходимыми условиями распознавания являются:

1. Исходный граф является связным
2. Максимальная степень вершины не больше 6 (иначе граф точно не укладывается в регулярную сетку размерности 3 и ниже)

## 2.2. Концепция схемы решения задачи нумерации

Шаг 1:

Выбор опоры - точки начала координат. Выбирается по максимальному числу соседствующих вершин.

Шаг 2:

Пронумеровать соседей, т.е. задать направления осей для сетки. Если дальнейшая нумерация не удалась, то изменяем их нумерацию.

Шаг 3:

Есть два случая, когда мы можем однозначно поставить индекс для вершины:

1. Два и более соседей имеют индекс
2. Один из соседей не имеет непронумерованных соседей кроме текущей вершины

Шаг 4:

На неоднозначных случаях рекурсивно перебираем все возможные индексы:

1. Ставим один из возможных индексов
2. Повторяем алгоритм
3. Если не удалось пронумеровать, то возвращаемся, меняем индекс и проходим заново

## 2.3. Описание алгоритма для одномерного случая

Шаг 1:

Проверяем, что две вершины графа имеют степень 1, а все остальные – 2

Шаг 2:

Вершине со степенью 1 присваиваем номер 1 и последовательно нумеруем соседей

## 2.4. Описание алгоритма для двумерного случая

Шаг 1:

Находим вершину старшей степени (максимальная степень – 4), нумеруем нулевыми значениями.

Шаг 2:

Нумеруем смежные вершины *(P)* по очереди.

Если дальнейшая нумерация не удалась, то пробуем поменять координаты вершин местами и запустить нумерацию заново.

До 24 вариантов нумераций *P*.

Шаг 3:

* Нумеруем смежные вершины от *P* по очереди.
* Однозначно разрешимые случаи:
  + Два и более соседей имеют индекс.
  + Один из соседей не имеет непронумерованных соседей кроме текущей вершины.

Шаг 4:

* Для остальных вершин рекурсивно перебираем все возможные индексы:
  + Ставим один из возможных индексов.
  + Повторяем алгоритм для смежных вершин.

Шаг 5:

* Если не удалось пронумеровать, то:
  + Возвращаемся к предыдущему неоднозначному случаю
  + Меняем индекс на следующий возможный
  + Запускаем дальнейшую нумерацию

## 2.5. Описание алгоритма для трехмерного случая

Шаг 1:

* Находим вершину старшей степени (максимальная степень – 6), нумеруем нулевыми значениями.

Шаг 2:

* Нумеруем смежные вершины *(P)* по очереди.
* Если дальнейшая нумерация не удалась, то пробуем поменять координаты вершин местами и запустить нумерацию заново.
* До 720 вариантов нумераций *P*.

Шаг 3:

* Нумеруем смежные вершины от *P* по очереди.
* Однозначно разрешимые случаи:
  + Два и более соседей имеют индекс.
  + Один из соседей не имеет непронумерованных соседей кроме текущей вершины.

Шаг 4:

* Для остальных вершин рекурсивно перебираем все возможные индексы:
  + Ставим один из возможных индексов.
  + Повторяем алгоритм для смежных вершин.

Шаг 5:

* Если не удалось пронумеровать, то:
  + Возвращаемся к предыдущему неоднозначному случаю
  + Меняем индекс на следующий возможный
  + Запускаем дальнейшую нумерацию

## 2.6. Вычислительная сложность алгоритмов распознавания и нумерации

Сложность алгоритма распознавания - это сложность алгоритма обхода в глубину для графа G (V, E):

O (|V| + |E|) = O (|E|)

Сложность алгоритма нумерации:

1. Для одномерного случая – O (|V|)
2. Для двумерного случая – O (4! \* (3!) ^ (|V| - 1)) = O (6 ^ |V|)
3. Для трехмерного случая – O (6! \* (5!) ^ (|V| - 1)) = O (120 ^ |V|)

# 3. Программная реализация

## 3.1. Требования к программе распознавания и нумерации сеток

В библиотеке должно быть реализовано:

* Функция быстрой проверки на регулярность поданного на вход графа (должна выполняться не более 5 секунд).
* Функция восстановления регулярной нумерации при регулярности графа (должна выполняться не более 5 минут).

В консольном приложении должно быть реализовано:

* Чтение графа из файла.
* Выполнение функции проверки на регулярность графа (вывод на экран времени работы функции).
* Выполнение функции восстановления регулярной нумерации (вывод на экран времени работы функции).
* Сохранение полученной регулярной нумерации графа в файл.

## 3.2. Требования к исходным данным

Исходный граф для консольного приложения должен удовлетворять условиям пункта 1.1 и быть представлен в формате METIS.

Описание формата: <http://people.sc.fsu.edu/~jburkardt/data/metis_graph/metis_graph.html>.

Исходные данные должны иметь корректное представление, иначе поведение программы не определено.

## 3.3. Требования к форматам представления результата

Выходными данными консольного приложения является вектор регулярной нумерации, представленный в JSON формате.

Имеет следующий вид:

{  
 "0": {  
 "Y": -1,  
 "X": 0  
 },  
 "1": {  
 "Y": 0,  
 "X": -1  
 }  
}

Это словарь, где в качестве ключа выступает номер вершины графа, а значение - соответствующие координаты регулярной сетки.

## 3.4. Структура программы

Программа состоит из следующих компонент:

1. Loader - предназначен для считывания данных графа из METIS формата;
2. JsonSerializer - предназначен для сериализации результатов нумерации в JSON формат;
3. ArgParser - парсер входных аргументов;
4. MeshRecovery\_Lib:
   1. Validator - проверяет исходный граф на необходимые условия регулярности и возвращает минимально возможную размерность регулярной сетки, в которую можно уложить граф;
   2. Numerator1D - пытается пронумеровать граф индексами регулярной сетки размерности 1 и возвращает ошибку, если не удалось;
   3. Numerator2D - пытается пронумеровать граф индексами регулярной сетки размерности 2 и возвращает ошибку, если не удалось;
   4. Numerator3D - пытается пронумеровать граф индексами регулярной сетки размерности 3 и возвращает ошибку, если не удалось.

## 3.5. Описание программы

Запуск приложения «MeshRecovery\_Console.exe» с двумя аргументами:

1. i (input) – путь до файла в формате METIS (\*.graph).
2. o (output) – необязательный аргумент, путь до файла для сохранения результата в формате JSON. По умолчанию результат сохраняется в одноименном файле рядом с исходным файлом.

О передаче некорректных аргументов информируют следующие сообщения:

* "*Please specify the path to graph file*" – не указан путь к файлу с исходными данными;
* "*File is not exist: {sourceFile}*" – не найден файл с исходными данными;
* "*Output file must have .json extension*" – указано неверное расширение файла для сохранения результатов.

После выполнения функции Validate (Numerate) выдается соответствующее сообщение о продолжительности работы следующего формата:

"*Function Validate (Numerate) finished work. Elapsed: (время выполнения в миллисекундах)*"

После успешного выполнения функции Numerate выдается полный путь до файла, куда был сохранен результат.

Сообщения об ошибках:

* "*Graph can not be numerate*d" - если граф не удалось пронумеровать;
* "*Can not serialize result*" - если не удалось сериализовать результат;
* "*Can not save result*" - если не удалось сохранить результат в файл.

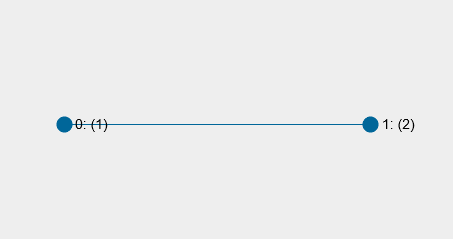
Внутренних проверок с проверкой топологии графа не проводится. Алгоритм работает из расчёта, что исходные данные представлены корректно.

# 4. Тестовый базис и верификация программы

## 4.1. Тестовые графы – одномерные сетки

line2.graph. 2 вершины. 1 ребро

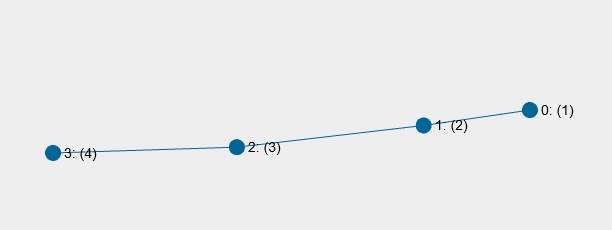
line2.graph



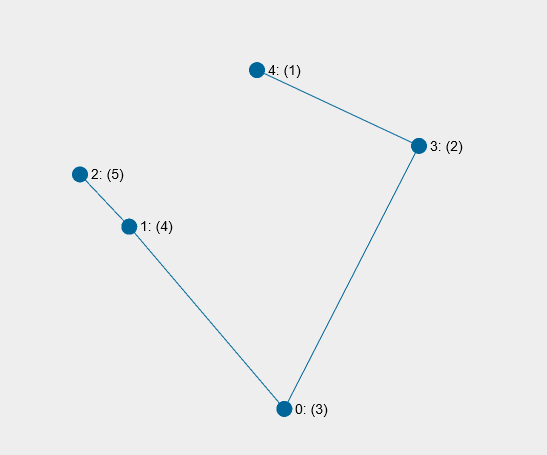
line3.graph. 3 вершины. 2 ребра



line4.graph. 4 вершины. 3 ребра

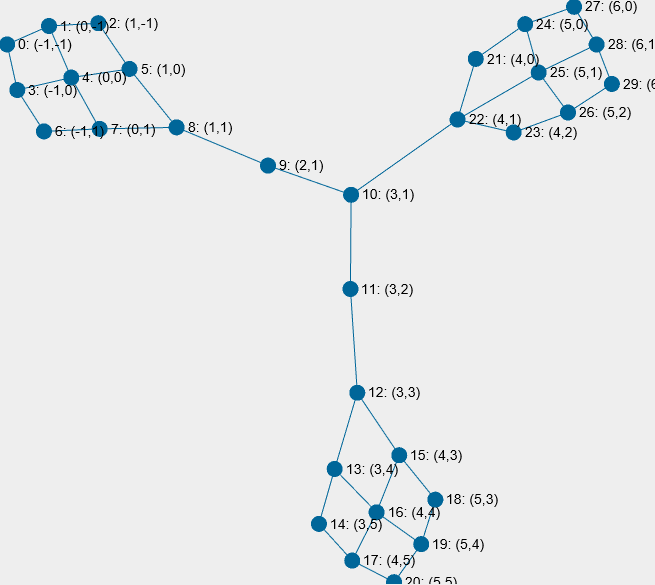


line\_mix.graph. 5 вершин. 4 ребра

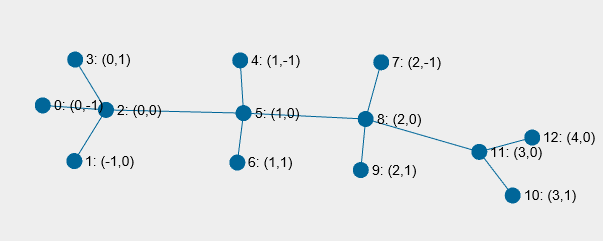


## 4.2. Тестовые графы – двумерные сетки

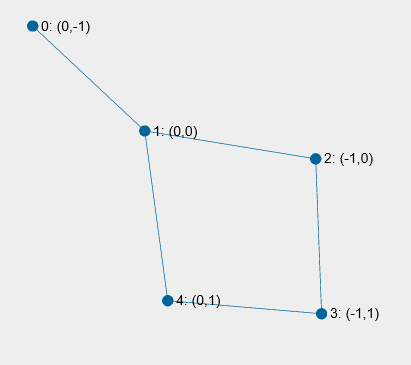
3\_domains.graph. 30 вершин. 41 ребро



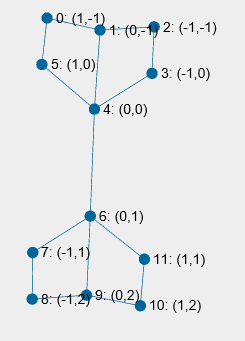
antenna.graph. 13 вершин. 12 рёбер



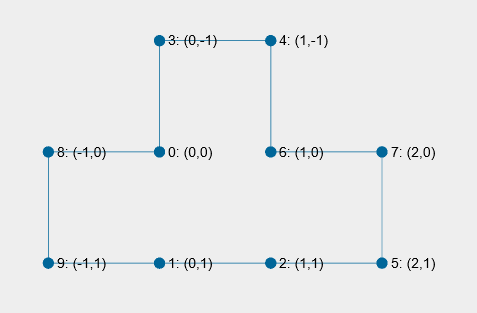
badtest\_k1.graph. 5 вершин. 5 рёбер



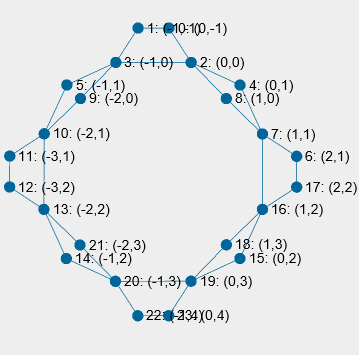
bridge.graph. 12 вершин. 15 рёбер



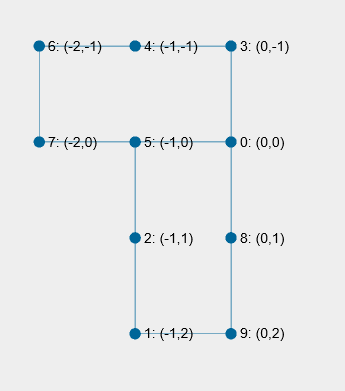
empty\_rectangle.graph. 10 вершин. 10 рёбер



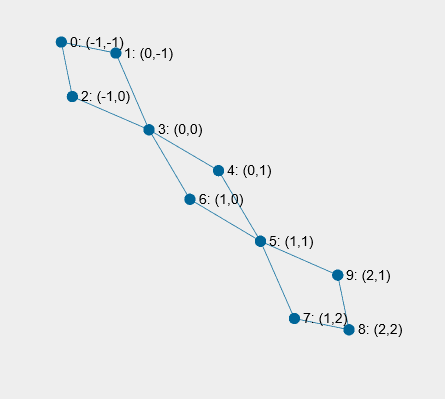
empty\_star.graph. 24 вершины. 32 ребра



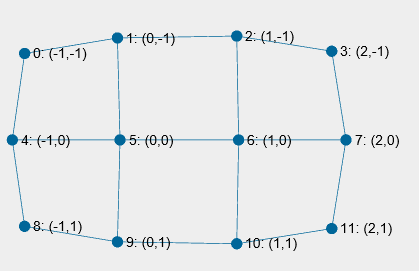
extra\_diag\_empty\_rectangle.graph. 10 вершин. 11 рёбер



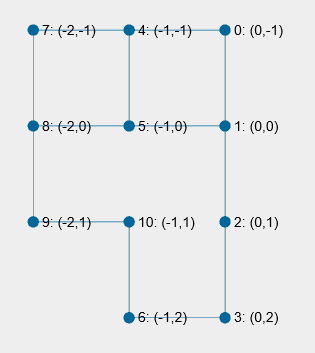
ladder.graph. 10 вершин. 12 рёбер



rectangle.graph. 12 вершин. 17 рёбер



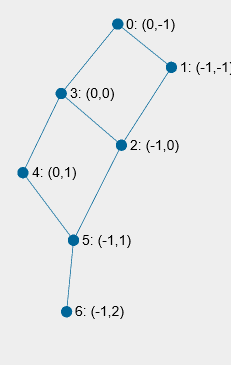
rectangle\_2.graph. 11 вершин. 13 рёбер



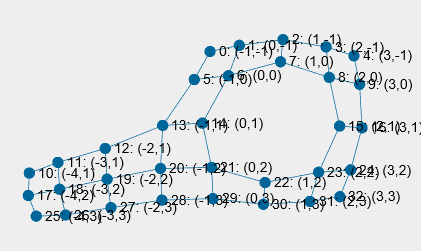
simple\_square.graph. 4 вершины. 4 ребра



square7.graph. 7 вершин. 8 рёбер



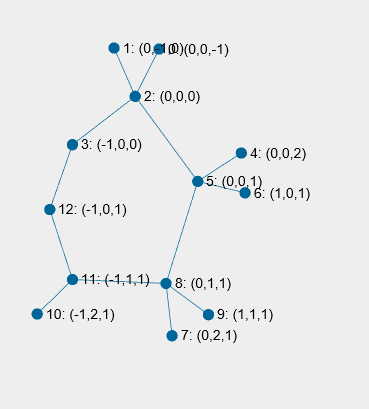
test1\_k2.graph. 33 вершины. 51 ребро



bad\_square.graph. 7 вершин. 9 рёбер

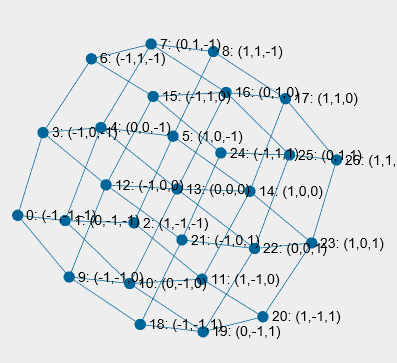


broken\_antenna.graph. 13 вершин. 13 рёбер

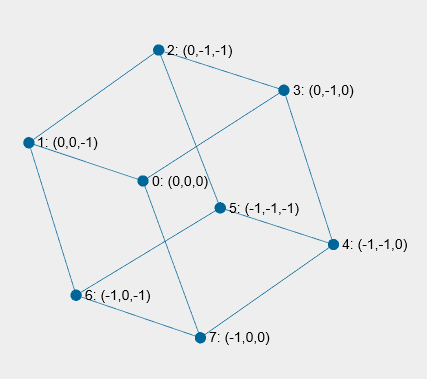


## 4.3. Тестовые графы – трехмерные сетки

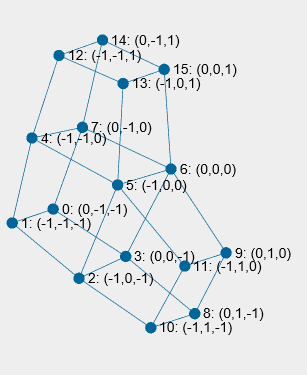
big\_cube.graph. 27 вершин. 54 ребра



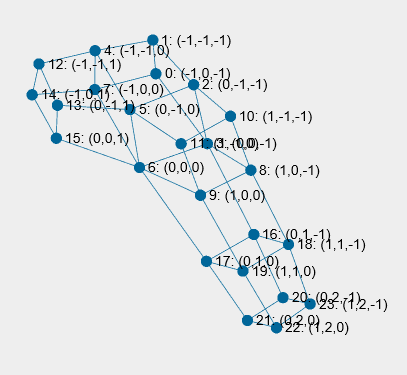
cube8.graph. 8 вершин. 12 рёбер



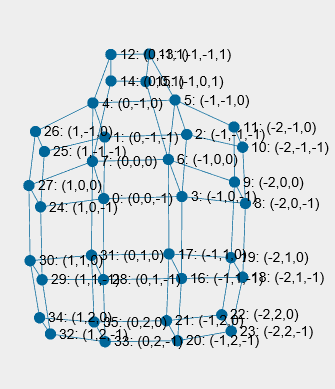
test2\_3D.graph. 16 вершин. 28 рёбер



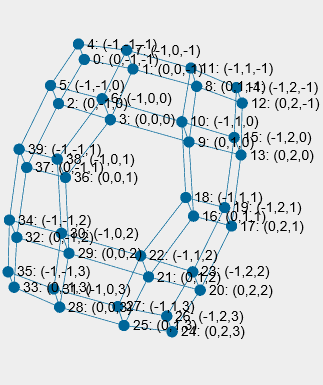
test3\_3D.graph. 24 вершины. 44 ребра



test4\_3D.graph. 36 вершин. 72 ребра

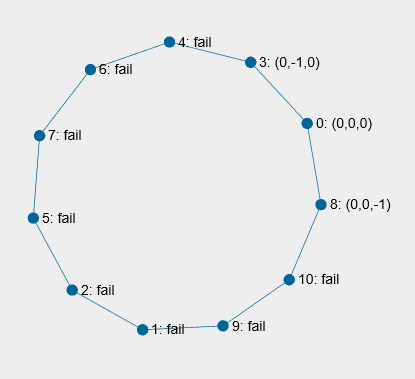


test5\_3D.graph. 40 вершин. 80 ребра

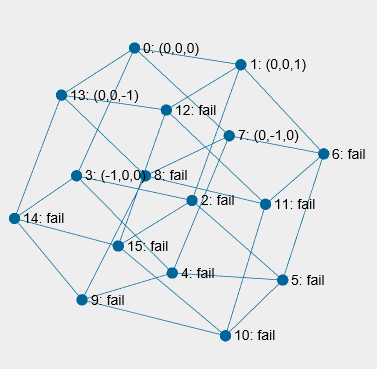


## 4.4. Тестовые графы – не сетки

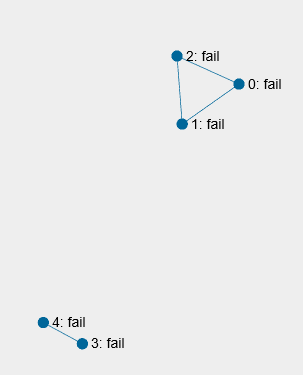
extra\_edge\_empty\_rectangle.graph. 11 вершин. 11 рёбер



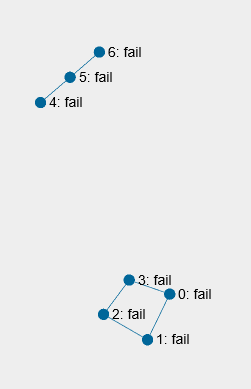
fake\_cube.graph. 16 вершин. 32 ребра



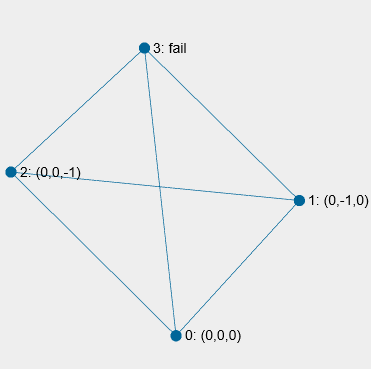
line\_bad\_1.graph. 5 вершин. 4 ребра



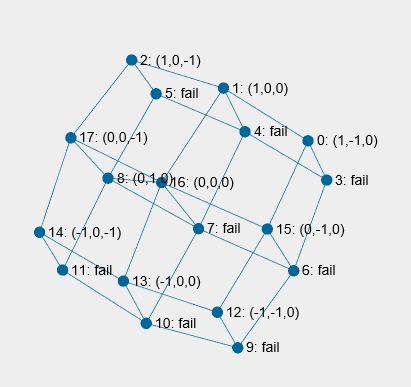
line\_bad\_8.graph. 7 вершин. 6 рёбер



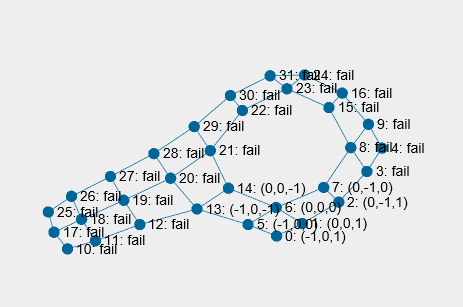
square.graph. 4 вершины. 4 рёбер



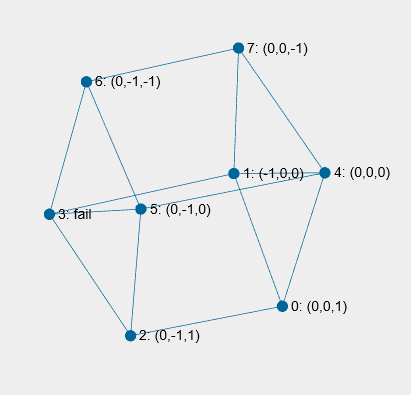
test1\_k3.graph. 18 вершин. 34 ребра



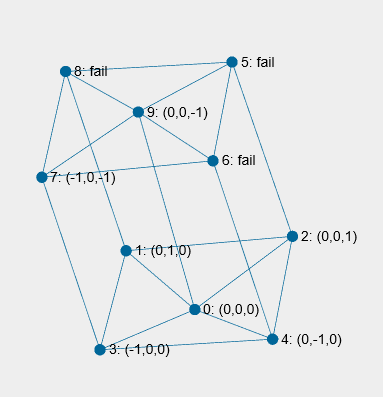
test2\_k2.graph. 32 вершины. 50 рёбер



test6\_3D\_error.graph. 8 вершин. 14 рёбер



test7\_3D\_error.graph. 10 вершин. 21 ребро



## 4.5. Тестовые графы для тестов производительности

fe\_3elt.graph. 4720 вершин. 13722 ребра

fe\_bracket.graph. 62631 вершин. 366559 рёбер

fe\_rotor.graph. 99617 вершин. 662431 ребро

ef\_body.graph. 45087 вершин. 163734 ребра

ef\_sphere.graph. 16386 вершин. 49152 ребра

fe\_tooth.graph. 78136 вершин. 452591 ребро

mdual.graph. 258569 вершин. 513132 ребра

grid\_150x200.graph. 30000 вершин. 59650 рёбер

# 5. Верификация и тестирование программы

## 5.1. Методика верификации

Подготовлена тестовая база, состоящая из примеров графов, в которой выделяются две группы:

1. Графы, которые возможно пронумеровать индексами регулярной сетки размерности от 1 до 3 (далее - хорошие графы).
2. Графы, которые нельзя пронумеровать индексами регулярной сетки (далее - плохие графы).

Для каждого примера из тестовой базы выполняются следующие шаги:

1. Запуск быстрой проверки графа на необходимые условия регулярности
2. Если граф распознан, то запускается алгоритм нумерации
3. Если нумерация построена успешно, то запускается проверка полученных индексов на корректность следующими условиями:
   1. Нет одинаковых индексов
   2. Индексы соседей у каждой вершины отличаются от ее индекса только в одной позиции и только на единицу

## 5.2. Результаты верификации

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Распознал граф | Построил нумерацию | Корректность нумерации |
| 3\_domains.graph | ИСТИНА | ИСТИНА | Верно |
| antenna.graph | ИСТИНА | ИСТИНА | Верно |
| badtest\_k1.graph | ИСТИНА | ИСТИНА | Верно |
| bad\_square.graph | ИСТИНА | ИСТИНА | Верно |
| big\_cube.graph | ИСТИНА | ИСТИНА | Верно |
| bridge.graph | ИСТИНА | ИСТИНА | Верно |
| broken\_antenna.graph | ИСТИНА | ИСТИНА | Верно |
| cube8.graph | ИСТИНА | ИСТИНА | Верно |
| empty\_rectangle.graph | ИСТИНА | ИСТИНА | Верно |
| empty\_star.graph | ИСТИНА | ИСТИНА | Верно |
| extra\_diag\_empty\_rectangle.graph | ИСТИНА | ИСТИНА | Верно |
| grid\_150x200.graph | ИСТИНА | ИСТИНА | Верно |
| ladder.graph | ИСТИНА | ИСТИНА | Верно |
| line2.graph | ИСТИНА | ИСТИНА | Верно |
| line3.graph | ИСТИНА | ИСТИНА | Верно |
| line4.graph | ИСТИНА | ИСТИНА | Верно |
| line\_mix.graph | ИСТИНА | ИСТИНА | Верно |
| rectangle.graph | ИСТИНА | ИСТИНА | Верно |
| rectangle\_2.graph | ИСТИНА | ИСТИНА | Верно |
| simple\_square.graph | ИСТИНА | ИСТИНА | Верно |
| square7.graph | ИСТИНА | ИСТИНА | Верно |
| test1\_3D.graph | ИСТИНА | ИСТИНА | Верно |
| test1\_k2.graph | ИСТИНА | ИСТИНА | Верно |
| test2\_3D.graph | ИСТИНА | ИСТИНА | Верно |
| test3\_3D.graph | ИСТИНА | ИСТИНА | Верно |
| test4\_3D.graph | ИСТИНА | ИСТИНА | Верно |
| test5\_3D.graph | ИСТИНА | ИСТИНА | Верно |
| ef\_body.graph | ЛОЖЬ | ЛОЖЬ | X |
| ef\_sphere.graph | ИСТИНА | ЛОЖЬ | X |
| extra\_edge\_empty\_rectangle.graph | ИСТИНА | ЛОЖЬ | X |
| fake\_cube.graph | ИСТИНА | ЛОЖЬ | X |
| fe\_3elt.graph | ЛОЖЬ | ЛОЖЬ | X |
| fe\_bracket.graph | ЛОЖЬ | ЛОЖЬ | X |
| fe\_rotor.graph | ЛОЖЬ | ЛОЖЬ | X |
| fe\_tooth.graph | ЛОЖЬ | ЛОЖЬ | X |
| line\_bad\_1.graph | ЛОЖЬ | ЛОЖЬ | X |
| line\_bad\_8.graph | ЛОЖЬ | ЛОЖЬ | X |
| mdual.graph | ИСТИНА | ЛОЖЬ | X |
| square.graph | ИСТИНА | ЛОЖЬ | X |
| test1\_k3.graph | ИСТИНА | ЛОЖЬ | X |
| test2\_k2.graph | ИСТИНА | ЛОЖЬ | X |
| test6\_3D\_error.graph | ИСТИНА | ЛОЖЬ | X |
| test7\_3D\_error.graph | ИСТИНА | ЛОЖЬ | X |

## Табл 5.1. Результаты верификации

## 5.3. Методика проведения тестов производительности

Для проведения тестов на производительность используются примеры с количеством вершин в графе от 4720 до 258569.

В ходе тестирования фиксируется рабочее время алгоритмов проверки и нумерации для каждого примера.

## 5.4. Результаты тестов производительности

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя файла | Кол-во вершин | Кол-во ребер | Корректность нумерации | Размерность | Время выполнения (ms) |
| grid\_150x200.graph | 30000 | 59650 | Верно | 2 | 309233 |
| ef\_body.graph | 45087 | 163734 | X | X | 3 |
| ef\_sphere.graph | 16386 | 49152 | X | X | 3432 |
| fe\_3elt.graph | 4720 | 13722 | X | X | 1 |
| fe\_bracket.graph | 62631 | 366559 | X | X | 29 |
| fe\_rotor.graph | 99617 | 662431 | X | X | 57 |
| fe\_tooth.graph | 78136 | 452591 | X | X | 46 |
| mdual.graph | 258569 | 513132 | X | X | 11570 |

## Табл 5.1. Результаты тестов производительности

## 5.5. Выводы по результатам верификации и тестирования программы

В ходе анализа сводных результатов верификации и тестирования производительности были сделаны следующие выводы:

1. Процедура быстрой проверки позволяет сразу говорить о невозможности нумерации, не запуская алгоритм нумерации, для некоторых плохих графов.
2. Скорость работы алгоритма нумерации напрямую зависит от топологии исходного графа.

# Заключение

1. Сделано:
   1. Написана библиотека, позволяющая:
      1. Проверить граф на необходимые условия регулярности;
      2. Восстановить геометрическую информацию для исходного графа.
   2. Написано консольное приложение, позволяющее:
      1. Считать исходный граф, представленный в METIS формате;
      2. Проверить граф на регулярность и восстановить нумерацию;
      3. Сохранить полученный результат в JSON формате.
   3. Создана тестовая инфраструктура с расширяемой базой.
2. Проблемы:
   1. Предложенное решение позволяет решать только некоторый класс задач;
   2. Время выполнения сильно зависит от топологии исходного графа.
3. Перспективы:
   1. Ускорение текущего алгоритма:
      1. Распараллеливание нумерации;
      2. Определение класса задач до начала нумерации;
   2. Исследование других подходов к решению задачи.