Титульник

[Введение. 3](#_Toc501268044)

[1. Математическая постановка задач распознавания и нумерации 4](#_Toc501268045)

[1.1. Исходные данные. 4](#_Toc501268046)

[1.2. Размерность в задаче распознавания и нумерации 4](#_Toc501268047)

[1.3. Математическая модель 4](#_Toc501268048)

[1.3.1. Одномерный случай 4](#_Toc501268049)

[1.3.2. Двумерный случай 4](#_Toc501268050)

[1.3.3. Трехмерный случай 4](#_Toc501268051)

[1.4. Критерии задачи нумерации 4](#_Toc501268052)

[2. Алгоритмы распознавания и нумерации 5](#_Toc501268053)

[2.1. Необходимые условия распознавания 5](#_Toc501268054)

[2.2. Концепция схемы решения задачи нумерации 5](#_Toc501268055)

[2.3. Описание алгоритма для одномерного случая 5](#_Toc501268056)

[2.4. Описание алгоритма для двумерного случая 5](#_Toc501268057)

[2.5. Описание алгоритма для трехмерного случая 6](#_Toc501268058)

[2.6. Вычислительная сложность алгоритмов распознавания и нумерации 7](#_Toc501268059)

[3. Программная реализация 8](#_Toc501268060)

[3.1. Требования к программе распознавания и нумерации сеток 8](#_Toc501268061)

[3.2. Требования к исходным данным 8](#_Toc501268062)

[3.3. Требования к форматам представления результата 8](#_Toc501268063)

[3.4. Структура программы 9](#_Toc501268064)

[3.5. Описание программы 9](#_Toc501268065)

[4. Тестовый базис и верификация программы 12](#_Toc501268066)

[4.1. Тестовые графы – одномерные сетки 12](#_Toc501268067)

[4.2. Тестовые графы – двумерные сетки 13](#_Toc501268068)

[4.3. Тестовые графы – трехмерные сетки 18](#_Toc501268069)

[4.4. Тестовые графы – не сетки 20](#_Toc501268070)

[4.5. Тестовые графы для тестов производительности 23](#_Toc501268071)

[5. Верификация и тестирование программы 24](#_Toc501268072)

[5.1. Методика верификации 24](#_Toc501268073)

[5.2. Результаты верификации 24](#_Toc501268074)

[5.3. Методика проведения тестов производительности 27](#_Toc501268075)

[5.4. Результаты тестов производительности 27](#_Toc501268076)

[5.5. Выводы по результатам верификации и тестирования программы 29](#_Toc501268077)

[Заключение. 31](#_Toc501268078)

# Введение.

Во многих областях используются расчетные (вычислительные) сетки при моделировании объектов.

При работе с крупными (точными) моделями возникает необходимость в их декомпозиции, но при этом теряется геометрическая информация вершин сетки.

В данном случае рассматривается регулярная сетка.

Регулярная сетка заданной размерности k состоит из узлов. Каждый узел имеет номер, состоящий из k компонент (). Каждый узел связан с соседними узлами. Регулярность означает, что:

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

Цель проекта – предложить возможное решение задачи о восстановлении геометрической информации для вершин графа в рамках регулярной сетки.

Задачи:

1. Написать библиотеку, которая позволяет:
   1. Проверить граф на соответствие необходимым условиям регулярности
   2. Восстановить нумерацию в случае регулярности графа
2. Написать консольное приложение, которое позволяет:
   1. Считать исходный граф, представленный в METIS формате
   2. Проверить граф на регулярность и восстановить нумерацию
   3. Сохранить результат в JSON формате
3. Создать тестовую инфраструктуру

# 1. Математическая постановка задач распознавания и нумерации

## 1.1. Исходные данные.

Неориентированный помеченный граф G = (V, E), заданный матрицей смежности:

пример:

Граф G:

## 1.2. Размерность в задаче распознавания и нумерации

k – размерность соответствующей регулярной сетки,

## 1.3. Математическая модель

1. Решение
   * – значение j-ой компоненты индекса регулярной сетки для i-ой вершины,
2. Ограничения
   * Граф G – связный

### 1.3.1. Одномерный случай

### 1.3.2. Двумерный случай

### 1.3.3. Трехмерный случай

## 1.4. Критерии задачи нумерации

# 2. Алгоритмы распознавания и нумерации

## 2.1. Необходимые условия распознавания

Функция Validate проверяет необходимые условия распознавания, такие как:

1. проверка на связность, где используется алгоритм пометок всех соседствующих вершин, и нахождение несвязных элементов
2. проверка максимальной степени вершины (если больше 6, то граф точно не укладывается в сетку размерности 3)

Также выполняет определение минимальной размерности, которая может быть применима для данной сетки.

## 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:
* Два и более соседей имеют индекс
* Один из соседей не имеет непронумерованных соседей кроме текущей вершины

Шаг 4:

* На неоднозначных случаях рекурсивно перебираем все возможные индексы:
* Ставим один из возможных индексов
* Повторяем алгоритм
* 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 из файла графа.
* Выполнение функции проверки на регулярность графа (вывод на экран времени работы функции).
* Выполнение функции восстановления регулярной нумерации (вывод на экран времени работы функции).
* Вывод полученной регулярной нумерации графа в файл.

## 3.2. Требования к исходным данным

Исходными данными консольного приложения является граф регулярной сетки, в котором вершины соответствуют узлам без нумерации, а рёбра связывают вершины – соседние узлы. Допустимые варианты размерности регулярной сетки 1, 2, 3.

Исходный граф для консольного приложения представлен в формате METIS (<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, в значение - соответствующие координаты регулярной сетки.

## 3.4. Структура программы

Программа состоит из следующих компонент:

1. Loader - предназначен для загрузки 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). (<http://people.sc.fsu.edu/~jburkardt/data/metis_graph/metis_graph.html>). Граф должен быть симметричный, не содержать петель;
2. O(output) – необязательный аргумент, путь до файла для сохранения результата в формате JSON. По умолчанию, если аргумент не указан, результат сохраниться 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 numerated" - 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.

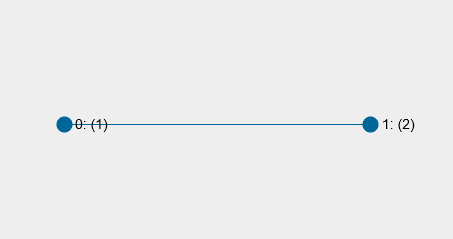
Внутренних проверок с проверкой топологии графа не проводится. Алгоритм работает из расчёта, что исходные данные представлены корректно.

# 4. Тестовый базис и верификация программы

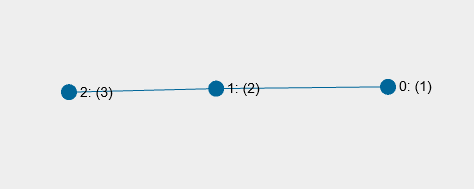
## 4.1. Тестовые графы – одномерные сетки

line2.graph. 2 вершины. 1 ребро

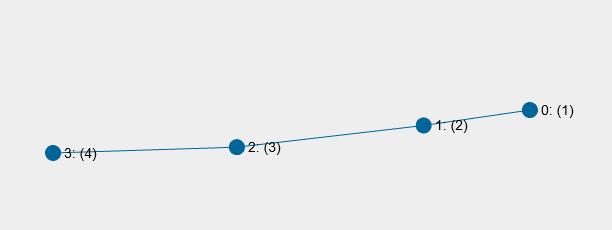
line2.graph



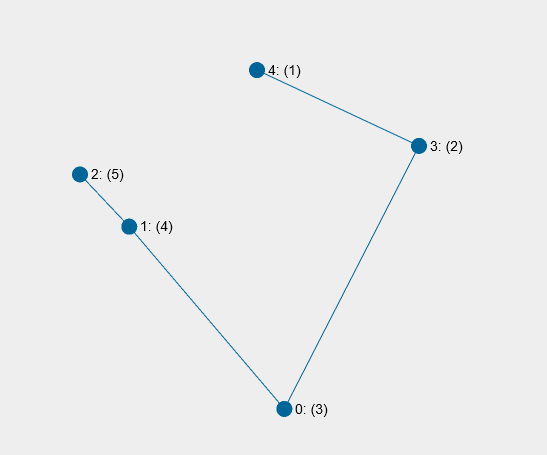
line3.graph. 3 вершины. 2 ребра



line4.graph. 4 вершины. 3 ребра

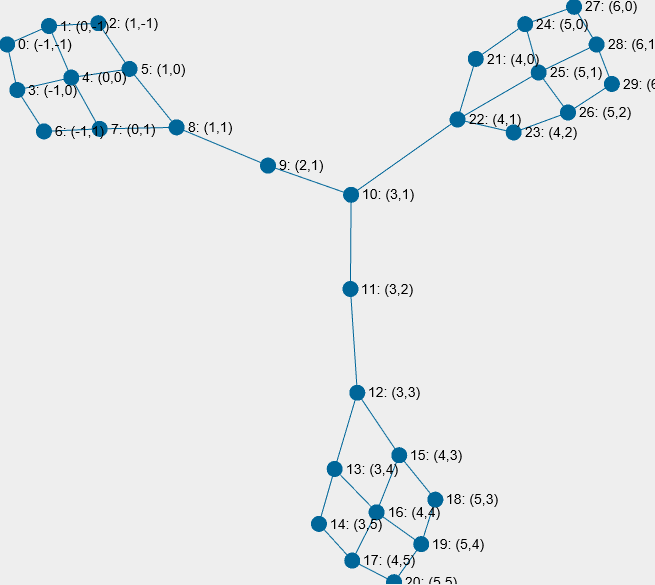


line\_mix.graph. 5 вершин. 4 ребра

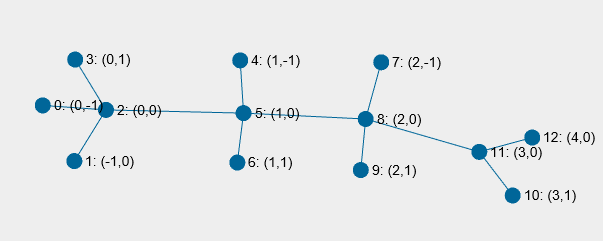


## 4.2. Тестовые графы – двумерные сетки

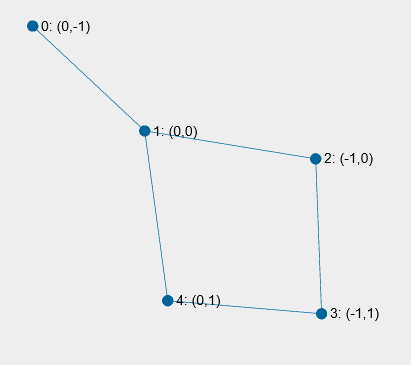
3\_domains.graph. 30 вершин. 41 ребро



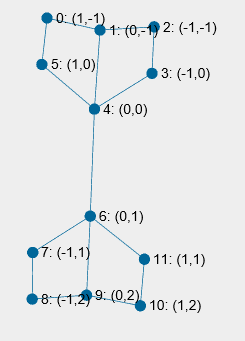
antenna.graph. 13 вершин. 12 рёбер



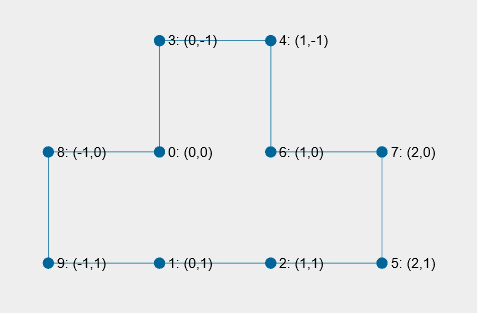
badtest\_k1.graph. 5 вершин. 5 рёбер



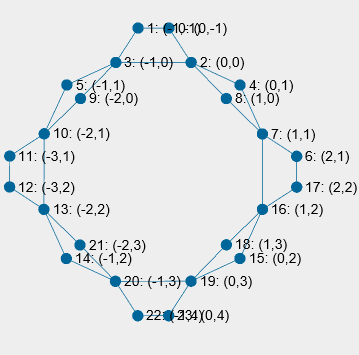
bridge.graph. 12 вершин. 15 рёбер



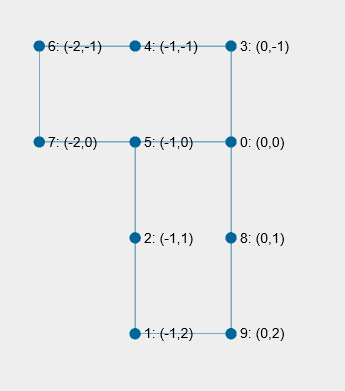
empty\_rectangle.graph. 10 вершин. 10 рёбер



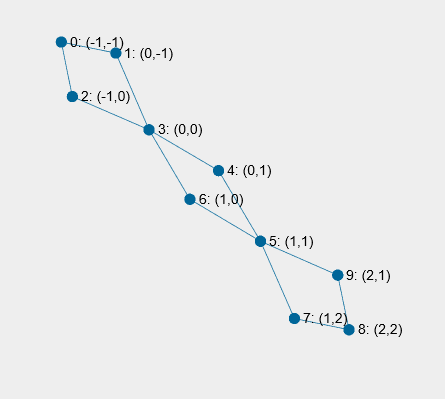
empty\_star.graph. 24 вершины. 32 ребра



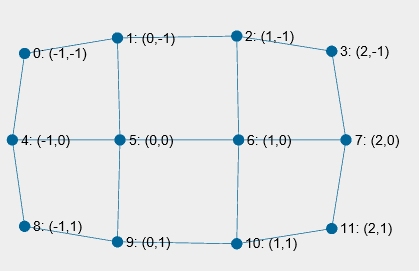
extra\_diag\_empty\_rectangle.graph. 10 вершин. 11 рёбер



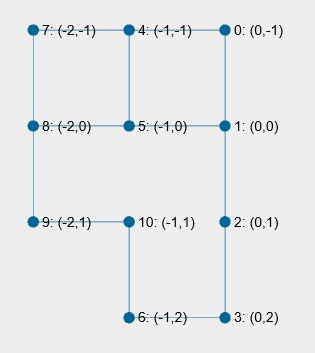
ladder.graph. 10 вершин. 12 рёбер



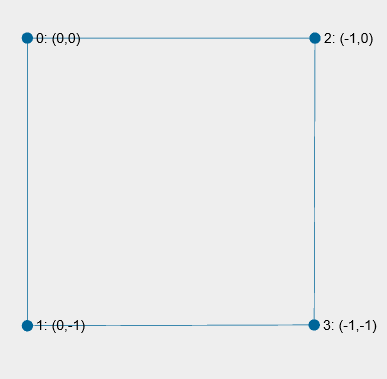
rectangle.graph. 12 вершин. 17 рёбер



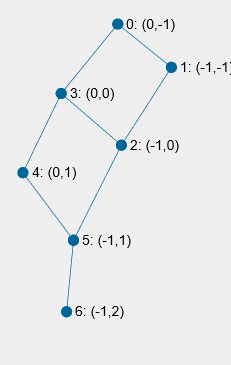
rectangle\_2.graph. 11 вершин. 13 рёбер



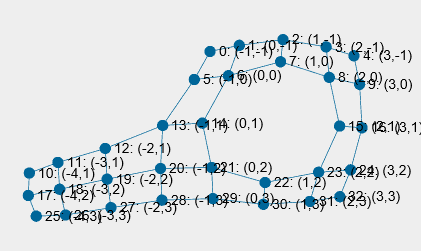
simple\_square.graph. 4 вершины. 4 ребра



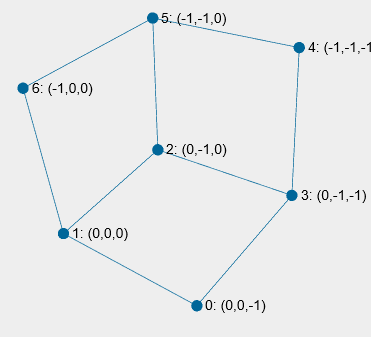
square7.graph. 7 вершин. 8 рёбер



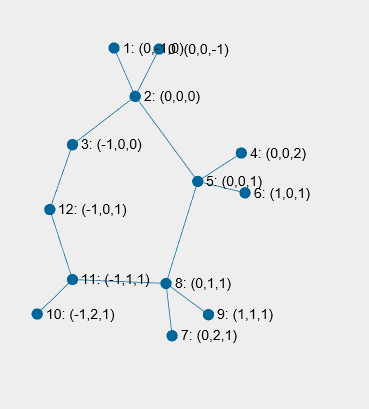
test1\_k2.graph. 33 вершины. 51 ребро



bad\_square.graph. 7 вершин. 9 рёбер

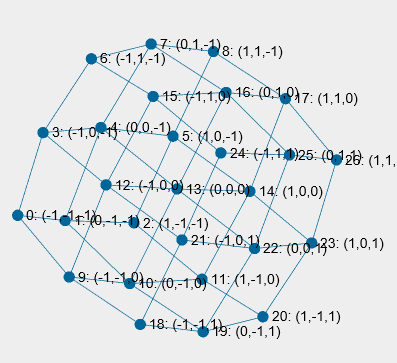


broken\_antenna.graph. 13 вершин. 13 рёбер

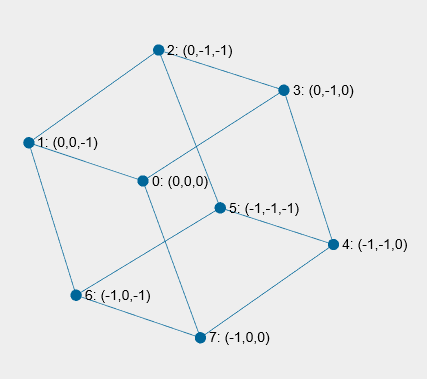


## 4.3. Тестовые графы – трехмерные сетки

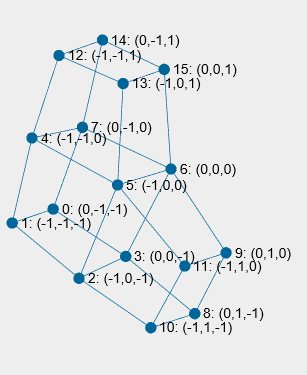
big\_cube.graph. 27 вершин. 54 ребра



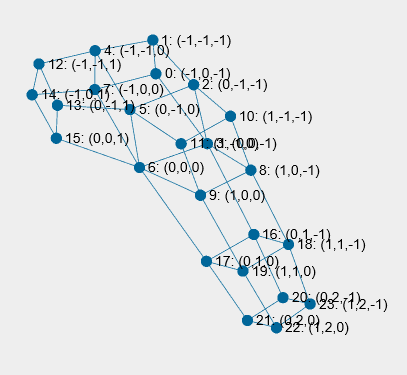
cube8.graph. 8 вершин. 12 рёбер



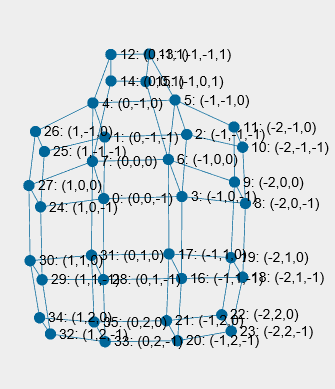
test2\_3D.graph. 16 вершин. 28 рёбер



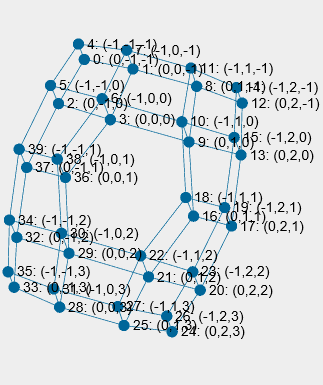
test3\_3D.graph. 24 вершины. 44 ребра



test4\_3D.graph. 36 вершин. 72 ребра

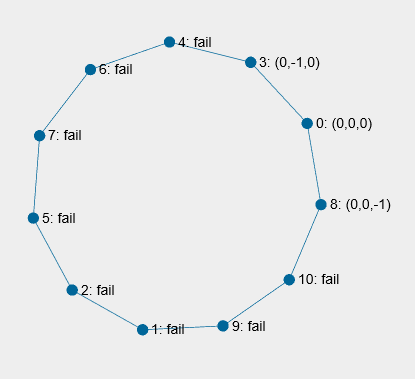


test5\_3D.graph. 40 вершин. 80 ребра

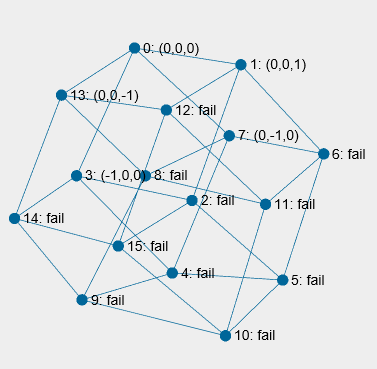


## 4.4. Тестовые графы – не сетки

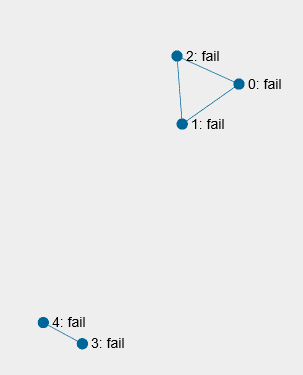
extra\_edge\_empty\_rectangle.graph. 11 вершин. 11 рёбер



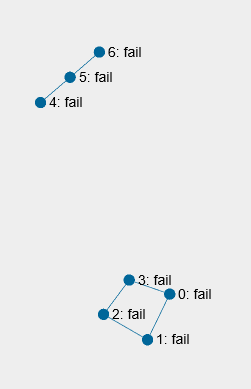
fake\_cube.graph. 16 вершин. 32 ребра



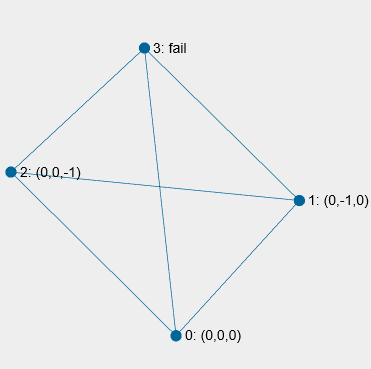
line\_bad\_1.graph. 5 вершин. 4 ребра



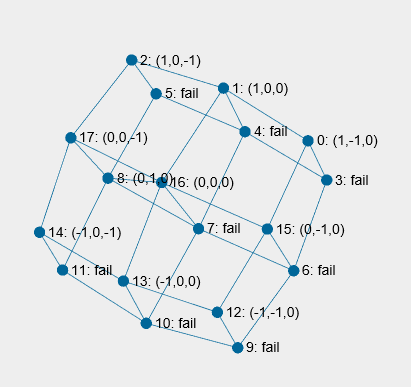
line\_bad\_8.graph. 7 вершин. 6 рёбер



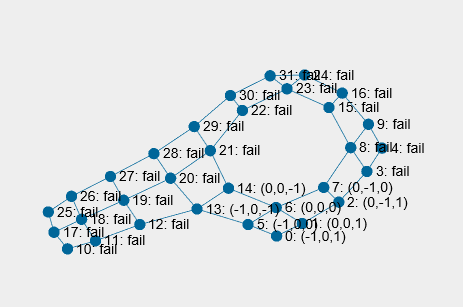
square.graph. 4 вершины. 4 рёбер



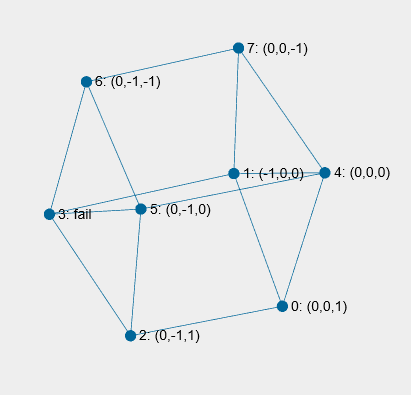
test1\_k3.graph. 18 вершин. 34 ребра



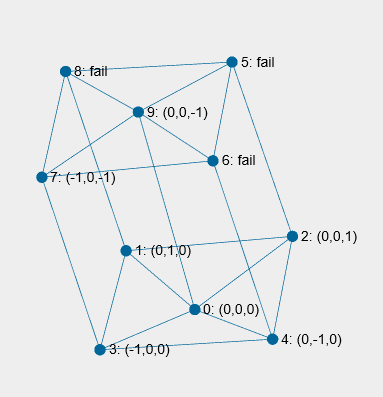
test2\_k2.graph. 32 вершины. 50 рёбер



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

Верификация заключается в:

1. Тестировании алгоритма на исходных данных, для которых заранее известен результат решения. В частности, были подготовлены хорошие и плохие исходные данные для размерности 1,2 и 3.
2. Проверке результата работы алгоритма на корректность. Для это проверяются следующие условия:
   1. Среди всех проиндексированных вершин графа нет двух и более одинаковых.
   2. Для каждой отдельно взятой вершины проверяются все её смежные соседи на предмет того, что индексы отличаются только в одной позиции и только на единицу.

## 5.2. Результаты верификации

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Кол-во вершин | Кол-во ребер | Распознал граф |
| rectangle\_2.graph | 11 | 13 | ИСТИНА |
| simple\_square.graph | 4 | 4 | ИСТИНА |
| square7.graph | 7 | 8 | ИСТИНА |
| test1\_3D.graph | 8 | 12 | ИСТИНА |
| test1\_k2.graph | 33 | 51 | ИСТИНА |
| test2\_3D.graph | 16 | 28 | ИСТИНА |
| test3\_3D.graph | 24 | 44 | ИСТИНА |
| test4\_3D.graph | 36 | 72 | ИСТИНА |
| test5\_3D.graph | 40 | 80 | ИСТИНА |
| ef\_body.graph | 45087 | 163734 | ЛОЖЬ |
| ef\_sphere.graph | 16386 | 49152 | ИСТИНА |
| extra\_edge\_empty\_rectangle.graph | 11 | 11 | ИСТИНА |
| fake\_cube.graph | 16 | 32 | ИСТИНА |
| fe\_3elt.graph | 4720 | 13722 | ЛОЖЬ |
| fe\_bracket.graph | 62631 | 366559 | ЛОЖЬ |
| fe\_rotor.graph | 99617 | 662431 | ЛОЖЬ |
| fe\_tooth.graph | 78136 | 452591 | ЛОЖЬ |
| line\_bad\_1.graph | 5 | 4 | ЛОЖЬ |
| line\_bad\_8.graph | 7 | 6 | ЛОЖЬ |
| mdual.graph | 258569 | 513132 | ИСТИНА |
| square.graph | 4 | 4 | ИСТИНА |
| test1\_k3.graph | 18 | 34 | ИСТИНА |
| test2\_k2.graph | 32 | 50 | ИСТИНА |
| test6\_3D\_error.graph | 8 | 14 | ИСТИНА |
| test7\_3D\_error.graph | 10 | 21 | ИСТИНА |
| rectangle\_2.graph | 11 | 13 | ИСТИНА |
| simple\_square.graph | 4 | 4 | ИСТИНА |
| square7.graph | 7 | 8 | ИСТИНА |
| test1\_3D.graph | 8 | 12 | ИСТИНА |
| test1\_k2.graph | 33 | 51 | ИСТИНА |
| test2\_3D.graph | 16 | 28 | ИСТИНА |
| test3\_3D.graph | 24 | 44 | ИСТИНА |
| test4\_3D.graph | 36 | 72 | ИСТИНА |
| test5\_3D.graph | 40 | 80 | ИСТИНА |
| ef\_body.graph | 45087 | 163734 | ЛОЖЬ |
| ef\_sphere.graph | 16386 | 49152 | ИСТИНА |
| extra\_edge\_empty\_rectangle.graph | 11 | 11 | ИСТИНА |
| fake\_cube.graph | 16 | 32 | ИСТИНА |
| fe\_3elt.graph | 4720 | 13722 | ЛОЖЬ |
| fe\_bracket.graph | 62631 | 366559 | ЛОЖЬ |
| fe\_rotor.graph | 99617 | 662431 | ЛОЖЬ |
| fe\_tooth.graph | 78136 | 452591 | ЛОЖЬ |
| line\_bad\_1.graph | 5 | 4 | ЛОЖЬ |
| line\_bad\_8.graph | 7 | 6 | ЛОЖЬ |
| mdual.graph | 258569 | 513132 | ИСТИНА |
| square.graph | 4 | 4 | ИСТИНА |
| test1\_k3.graph | 18 | 34 | ИСТИНА |
| test2\_k2.graph | 32 | 50 | ИСТИНА |
| test6\_3D\_error.graph | 8 | 14 | ИСТИНА |
| test7\_3D\_error.graph | 10 | 21 | ИСТИНА |

## 5.3. Методика проведения тестов производительности

Для проведения тестов на производительность были подготовлены корректные и некорректные исходные данные с количеством вершин в графе от 4720 до 258569. В ходе теста фиксируется рабочее время алгоритма для текущих исходных данных и, если граф успешно нумеруется, результирующие индексы проверяются на корректность.

## 5.4. Результаты тестов производительности

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя файла | Кол-во вершин | Кол-во ребер | Построил нумерацию | Корректность нумерации | Время выполнения (ms) |
| 3\_domains.graph | 30 | 41 | ИСТИНА | Верно | 0 |
| antenna.graph | 13 | 12 | ИСТИНА | Верно | 0 |
| badtest\_k1.graph | 5 | 5 | ИСТИНА | Верно | 0 |
| bad\_square.graph | 7 | 9 | ИСТИНА | Верно | 0 |
| big\_cube.graph | 27 | 54 | ИСТИНА | Верно | 0 |
| bridge.graph | 12 | 15 | ИСТИНА | Верно | 0 |
| broken\_antenna.graph | 13 | 13 | ИСТИНА | Верно | 2 |
| cube8.graph | 8 | 12 | ИСТИНА | Верно | 0 |
| empty\_rectangle.graph | 10 | 10 | ИСТИНА | Верно | 0 |
| empty\_star.graph | 24 | 32 | ИСТИНА | Верно | 4 |
| extra\_diag\_empty\_rectangle.graph | 10 | 11 | ИСТИНА | Верно | 0 |
| grid\_150x200.graph | 30000 | 59650 | ИСТИНА | Верно | 309233 |
| ladder.graph | 10 | 12 | ИСТИНА | Верно | 0 |
| line2.graph | 2 | 1 | ИСТИНА | Верно | 0 |
| line3.graph | 3 | 2 | ИСТИНА | Верно | 0 |
| line4.graph | 4 | 3 | ИСТИНА | Верно | 0 |
| line\_mix.graph | 5 | 4 | ИСТИНА | Верно | 0 |
| rectangle.graph | 12 | 17 | ИСТИНА | Верно | 0 |
| rectangle\_2.graph | 11 | 13 | ИСТИНА | Верно | 0 |
| simple\_square.graph | 4 | 4 | ИСТИНА | Верно | 0 |
| square7.graph | 7 | 8 | ИСТИНА | Верно | 0 |
| test1\_3D.graph | 8 | 12 | ИСТИНА | Верно | 0 |
| test1\_k2.graph | 33 | 51 | ИСТИНА | Верно | 0 |
| test2\_3D.graph | 16 | 28 | ИСТИНА | Верно | 0 |
| test3\_3D.graph | 24 | 44 | ИСТИНА | Верно | 2 |
| test4\_3D.graph | 36 | 72 | ИСТИНА | Верно | 1 |
| test5\_3D.graph | 40 | 80 | ИСТИНА | Верно | 1 |
| ef\_body.graph | 45087 | 163734 | ЛОЖЬ |  | 3 |
| ef\_sphere.graph | 16386 | 49152 | ЛОЖЬ |  | 3432 |
| extra\_edge\_empty\_rectangle.graph | 11 | 11 | ЛОЖЬ |  | 1879 |
| fake\_cube.graph | 16 | 32 | ЛОЖЬ |  | 1 |
| fe\_3elt.graph | 4720 | 13722 | ЛОЖЬ |  | 1 |
| fe\_bracket.graph | 62631 | 366559 | ЛОЖЬ |  | 29 |
| fe\_rotor.graph | 99617 | 662431 | ЛОЖЬ |  | 57 |
| fe\_tooth.graph | 78136 | 452591 | ЛОЖЬ |  | 46 |
| line\_bad\_1.graph | 5 | 4 | ЛОЖЬ |  | 0 |
| line\_bad\_8.graph | 7 | 6 | ЛОЖЬ |  | 0 |
| mdual.graph | 258569 | 513132 | ЛОЖЬ |  | 11570 |
| square.graph | 4 | 4 | ЛОЖЬ |  | 0 |
| test1\_k3.graph | 18 | 34 | ЛОЖЬ |  | 11 |
| test2\_k2.graph | 32 | 50 | ЛОЖЬ |  | 373 |
| test6\_3D\_error.graph | 8 | 14 | ЛОЖЬ |  | 1 |
| test7\_3D\_error.graph | 10 | 21 | ЛОЖЬ |  | 3 |

## 5.5. Выводы по результатам верификации и тестирования программы

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

1. Отсутствие некорректных тестов, для которых процесс нумерации закончился успешно, свидетельствует о том, что процесс валидации позволяет сэкономить время, не запуская алгоритм нумерации для такого графа.
2. Необходимым условием правильной работы алгоритма нумерации является корректность исходных данных. В противном случае есть возможность получения неверного результата.
3. Скорость работы алгоритма нумерации напрямую зависит от его размерности и количества вершин, а также от топологии 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