Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет**

**имени первого Президента России Б. Н. Ельцина»**

**Институт естественных наук и математики**

**Порождение всех реализаций графического разбиения с точностью до изоморфизма**

Курсовая работа

студента группы КБ-301

Махаева С.В.

Научный руководитель:

профессор кафедры алгебры и

дискретной математики

ИЕНиМ УрФУ,

д.ф.-м.н.

Баранский В.А.

Екатеринбург

2017

**ОГЛАВЛЕНИЕ**

[**Введение** 2](#_Toc481108005)

[**1.** **Описание программы** 3](#_Toc481108006)

[**2.** **Порождение одной произвольной реализации разбиения** 4](#_Toc481108007)

[**3.** **Операция своп** 4](#_Toc481108008)

[**4.** **Изоморфизм разбиений** 4](#_Toc481108009)

[**Заключение** 4](#_Toc481108010)

[**Список литературы** 4](#_Toc481108011)

# **Введение**

Одним из активно развивающихся направлений в современной теории графов являются графические разбиения. Для начала дадим определение графического разбиения. Рассмотрим обыкновенный граф *G=(V,E)*. Под графическим разбиением понимают последовательность *(d1,d2,…,dn)*, где *dk*∈ℕ - степень *k*-ой вершины графа *G(k=1,2,…,n)*. Заметим, что такая последовательность задаёт множество графов, ей удовлетворяющих, то есть графов, в которых степень вершины *vi∈V* равна *di (i=1,2,…,n)* для некоторой нумерации вершин из *V*. Такие графы мы будем называть реализациями данного графического разбиения. А разбиения, у которых есть хотя бы одна реализация, будем называть графичными. Кроме того, сразу можно отметить, что в классе графов, удовлетворяющих последовательности *L*, могут возникнуть пары изоморфных графов. Наша задача заключается в том, чтобы по данному графическому разбиению построить все неизоморфные её реализации.

Для решения данной задачи в рамках работы была создана программа, которая по заданному графическому разбиению строит все её реализации с точностью до изоморфизма. Описание этой программы содержится в первом параграфе.

Кроме этого, работа включает в себя три параграфа, описывающих последовательно все этапы решения поставленной задачи. Реализованный в рамках исследования алгоритм организует свою работу поэтапно, выделяя подзадачи и разрешая их. В работе эти этапы рассмотрены независимо. Во втором параграфе описано получение одной произвольной реализации данного разбиения. Третий параграф содержит описание операции своп, посредством которой алгоритм порождает все реализации по одной из них. В четвёртом параграфе затронута проблема изоморфизма двух различных разбиений.

1. **Описание программы**

Программа написана для интерпретатора python 3.5 и находится в открытом доступе по ссылке: <https://github.com/SemyonMakhaev/graphical-sequences>.

В реализации использованы библиотеки *networkx* для структуры графа и *matplotlib.pyplot* для рисования графов, а также для построения графиков.

Использование: *python3 realizations.py [-h] [-dir directory] [-d] degree [degree …]*. Программа создаёт директорию, куда помещает изображения построенных графов в формате *png*.

Позиционные аргументы:

* *degree* – Степень очередной вершины графического разбиения.

Опциональные аргументы:

* *-h, --help* – Вывод справки по использованию.
* *-dir directory, --directory directory* – Директория для вывода картинок.
* *-d, --debug* – Запуск дополнительных проверок.

Реализация основного алгоритма содержится в файле *realizations.py*.

Также для данного решения написаны тесты, они находятся в файле *tests.py*. Запуск тестирующей утилиты: *python3 tests.py*.

Кроме того, для измерения эффективности алгоритма был написан модуль *measure.py*. Он замеряет время работы программы для входных данных заданной величины и строит график, который помещает в директорию *measures*. Запуск: *python3 measures.py [-h] [-s] [length]*.

Позиционные аргументы:

* *length* – Максимальная длина графического разбиения. По умолчанию – *10*. Утилита измерит время работы алгоритма на последовательностях длин от *1* до *length*.

Опциональные аргументы:

* *-h, --help* – Вывод справки по использованию.
* *-s, --save* – Сохранять ли график в файл. Если аргумент не передан, график будет построен и выведен в отдельном окне, где его можно будет сохранить или закрыть.

1. **Порождение одной произвольной реализации разбиения**

Итак, предположим, что нам дано некоторое графическое разбиение. Рассмотрим алгоритм построения одной реализации этого разбиения. Сперва в граф добавим столько вершин, сколько элементов в последовательности. Для каждого элемента запоминаем его позицию в разбиении.

Цикл по *i* от *1* до длины разбиения. Если первый его элемент ≤ *0*, то выходим из цикла. Добавляем ребро между вершинами *0* и *i*, вычитаем по *1* из степеней с номерами *0* и *i*. На каждом шаге осуществляем сортировку последовательности по убыванию элементов (степеней). В итоге получаем реализацию исходного графического разбиения.

Отметим также, что в случае если по завершению основного цикла хотя бы один элемент последовательности не равен нулю, то разбиение не является графичным.

Приведём листинг метода *get\_realization* из исходного кода программы, который принимает на вход графическое разбиение, строит одну произвольную его реализацию, используя вышеописанный алгоритм, и возвращает полученный граф.

def get\_realization(sequence):

"""Порождает одну реализацию данного разбиения."""

numbered\_sequence = list(sequence)

graph = networkx.Graph()

for i in range(len(sequence)):

numbered\_sequence[i] = [sequence[i], i]

if sequence[i] == 0:

graph.add\_node(i)

for \_ in range(len(numbered\_sequence)):

for idx in range(1, len(numbered\_sequence)):

if numbered\_sequence[0][0] <= 0:

break

if numbered\_sequence[idx][0] > 0 and not graph.has\_edge( \

numbered\_sequence[idx][1], numbered\_sequence[0][1]):

numbered\_sequence[0][0] -= 1

numbered\_sequence[idx][0] -= 1

graph.add\_edge(numbered\_sequence[0][1], numbered\_sequence[idx][1])

numbered\_sequence.sort(key=lambda items: items[0], reverse=True)

for items in numbered\_sequence:

if items[0] != 0:

error('Разбиние не графично')

sys.exit(0)

return graph

1. **Операция своп**
2. **Изоморфизм разбиений**

**Заключение**

**Список литературы**

1. Peter L. Erdos, Zoltan Kiraly, Istvan Miklos – «On the swap-distances of different realizations of a graphical degree sequence».
2. Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И. – «Лекции по теории графов».