**Московский авиационный институт**

**(национальный исследовательский университет)**

**Институт №8 «Компьютерные науки и прикладная математика»**

**Кафедра 806 «Вычислительная математика и программирование»**

**КУРСОВОЙ ПРОЕКТ**

**по курсу "Дискретная математика"**

**II семестр**

**на тему «Эйлеровы и гамильтоновы пути (циклы)»**

**Студент: Ахметшин Б.Р.**

**Группа: М8О-103Б-22, № 2**

**Руководитель: Яшина Н. П., доцент 805 кафедры**

Contents

[Теоретические сведения 3](#__RefHeading___Toc644_1155833029)

[Описание алгоритма 4](#__RefHeading___Toc646_1155833029)

[Программная реализация 9](#__RefHeading___Toc648_1155833029)

[Практическое применение 19](#__RefHeading___Toc650_1155833029)

[Источники и полезные ссылки: 20](#__RefHeading___Toc652_1155833029)

# Теоретические сведения

Задан произвольный неориентированный псевдограф, требуется отыскать эйлеровы и гамильтоновы пути и циклы, если таковые имеются. (Далее речь будет идти только о неориентированных псевдографах)

Пусть *G* — произвольный псевдограф. Путь (цикл) в *G* называется эйлеровым, если по каждому ребру *G* он проходит ровно один раз, и гамильтоновым, если он проходит через каждую вершину *G* единожды.

Псевдограф, содержащий эйлеровый цикл называется эйлеровым, а содержащий эйлеровый путь — полуэйлеровым. Аналогично определяется гамильтоновый и полугамильтоновый псевдографы. Очевидно, что эйлеровым и/или гамильтоновым путем (циклом) могут обладать лишь связные псевдографы, что далее будет предполагаться в формулировках.

Теорема 1: Критерий существования эйлерового цикла.

Связный псевдограф *G* содержит эйлеровый цикл тогда и только тогда, когда степени всех вершин *G* четны.

Теорема 2: Критерий существования эйлерового пути.

Связный псевдограф *G* содержит эйлеровый путь тогда и только тогда, когда вершин с нечетной степенью либо два, либо ноль.

Критерии и эффективные алгоритмы гамильтонового пути и цикла сложны и не приведены в методических материалах, поэтому в своей работе я принял решение искать гамильтоновы пути и циклы только в псевдографах, удовлетворяющих условие следующей теоремы, потому как в таких графах их поиск выполняется за полиномиальное время.

Теорема 3: Условие Дирака.

Связный граф *G* обладает гамильтоновым циклом, если n ≥3, δ ≥ n/2, где δ — наименьшая степень вершины *G.*

Очевидно, что Теорема 3 верна также и для псевдографов. Из Теоремы 3 следует, что связный псевдограф, удовлетворяющий её условию, также содержит гамильтоновый путь:

Добавим к *G* вершину R и ребра, соединяющие R со всеми остальными вершинами *G.* Т.к. получившийся граф удовлетворяет условию Т3, в нем есть гамильтоновый цикл. Вычленив вершину R из цикла, получим гамильтоновый путь.

# Описание алгоритма

**Поиск эйлерового пути.**

Если данный псевдограф *G* удовлетворяет условию Т2, следующий алгоритм находит его:

**Псевдокод:**

St — стек;

в St кладём любую вершину (стартовая вершина);

пока St не пустой

пусть V - значение на вершине St;

если степень(V) = 0, то

добавляем V к ответу;

снимаем V с вершины St;

иначе

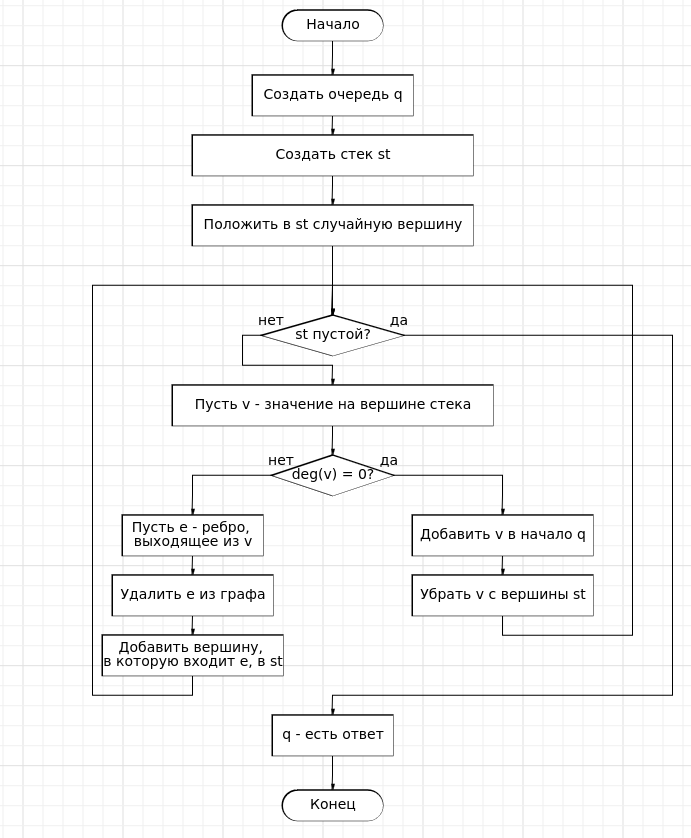
находим любое ребро, выходящее из V;

удаляем его из графа;

второй конец этого ребра кладём в St;

При условии, что каждое действие (на строчке псевдокода) выполняется за O(1), сложность алгоритма — O(m), где m — количество ребер псевдографа *G.*

**Блок-схема:**



**Поиск эйлерового цикла.**

Пусть псевдограф *G* удовлетворяет условию Т1, тогда для нахождения эйлерового цикла достаточно вычленить из *G* произвольное ребро, тогда *G* будет удовлетворять условию Т2, следовательно, в нем найдется эйлеровый путь. Если начать его поиск с одной из вершин изъятого ребра и после дополнить другой, получится эйлеровый цикл. Сложность O(m).

**Поиск гамильтонового цикла.**

Из доказательства Т3 следует алгоритм нахождения гамильтонового цикла. Итак, «Поступим следующим образом: заведем очередь и положим в нее все вершины нашего графа (не важно в каком порядке). Пусть *n* — количество вершин псевдографа. Тогда *n(n - 1)* раз будем делать следующую операцию:

* Пусть *v1*— это голова очереди, *v2* — следующая за ней вершина и так далее. Если между первой и второй вершиной в очереди есть ребро в графе *G*, то перемещаем первую вершину в конец очереди и переходим к следующей итерации.
* Если между первой и второй вершиной в очереди ребра нет, то найдем вершину *vi* где *i > 2*, такую что, ребра *v1vi,v2vi+1*∈*E* (так как у нас для графа выполнена [теорема Дирака](https://neerc.ifmo.ru/wiki/index.php?title=Теорема_Дирака), то такая вершина обязательно найдется). После чего поменяем в очереди местами вершины *v2* и *vi, v3* и *vi-1, v2+j* и *vi-j*, и так далее, пока *2 + j < i - j* (то есть *j* пробегает все значения от 0 до значения заданного неравенством). Теперь у нас появилось ребро между первой и второй вершинами в очереди (теперь вторая вершина, это та, которая была до разворота на *i-*йпозиции), а также, гарантированно существует ребро между *i-*йи (*i+1)-*й вершинами очереди. После этого, так же как и в первом случае, оправляем первую вершину в конец очереди.

Таким образом после *n* итераций, мы получаем последовательность (вершины лежащие в очереди), где любые 2 соседние вершины соединены ребром, все вершины графа находятся в этой последовательности, и более того, каждая ровно один раз, а также существует ребро между последней и первой вершинами очереди, а это и значит, что мы решили поставленную задачу.» - описание алгоритма нахождения гамильтонова цикла в условиях теорем Дирака и Оре поисточнику **[3]**. Сложность: O(n(n-1)).

**Псевдокод:**

q - очередь;

добавить в q вершины G в произвольном порядке;

Повторить n(n - 1) раз:

если (q0, q1) ∉ E:

i := 2;

Пока (q0, qi) ∉ E или (q1, qi+1) ∉ E:

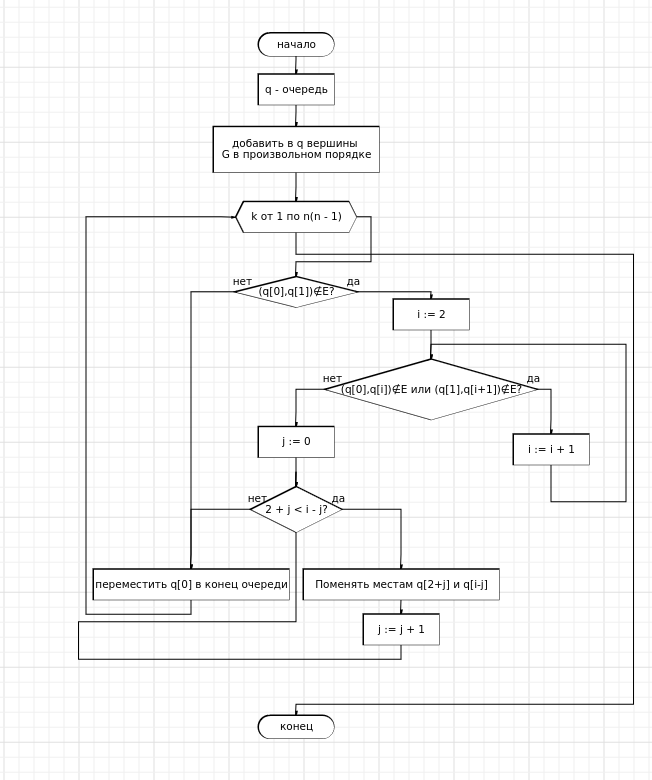
i = i + 1;

Поменять местами все пары (q2+j,qi-j) вершин \

в q такие, что 2 + j < i - j;

Переместить первую вершину в q в конец очереди

**Блок-схема:**



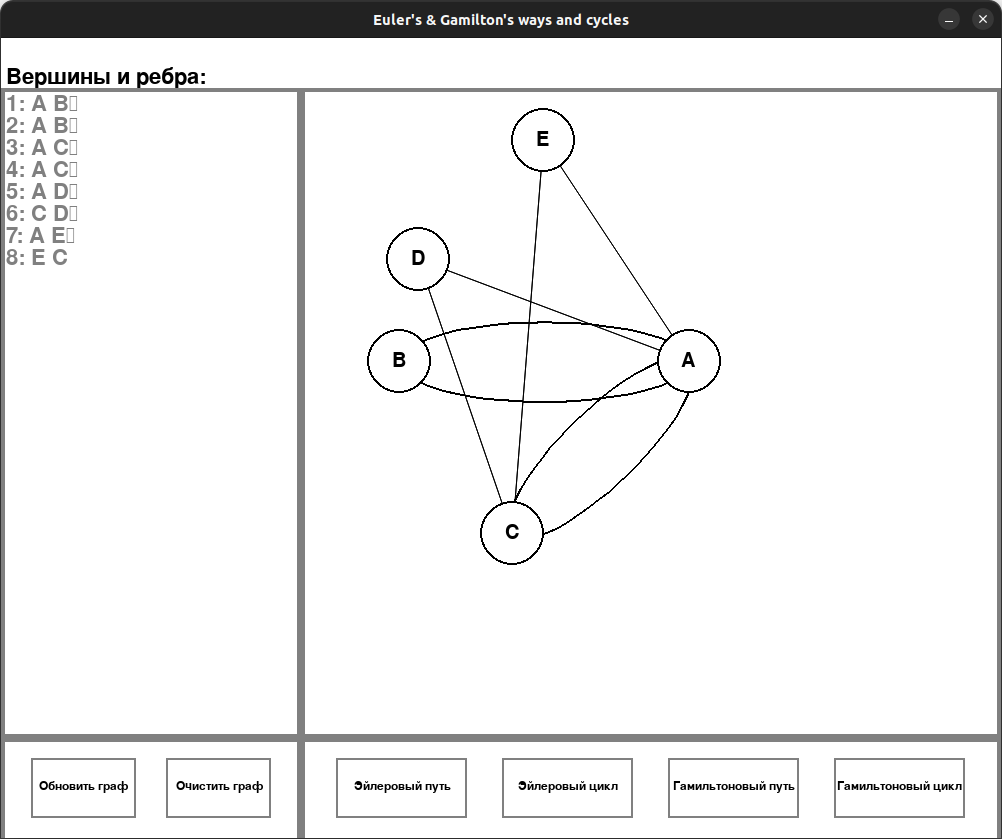
**Поиск гамильтонового пути.**

В конце пункта Теоретических сведений было указание для данного алгоритма. Еще раз: добавляем к псевдографу *G,* удовлетворяющему Т3, вершину *R,* соединенную ребрами со всеми остальными вершинами, получаем *G’.* Запустив предыдущий алгоритм, получим гамильтоновый цикл с вершиной *R.* Вычленив эту вершину из цикла, получим либо два гамильтоновых пути на подграфах *G,* тогда, совместив их начало и конец (за счет цикличности найдется ребро их соединяющее), получим гамильтоновый путь на *G,* либо один на подграфе *G,* совпадающем с ним самим, т. е. искомый путь. Сложность O(n(n - 1)).

# Программная реализация

**Описание программы**

Программа и себя представляет графическое приложение, разделенное на окна.



Левое верхнее окно — поле для ввода вершин и ребер графа, допустимо указывать вершины в отдельной строке, тогда, если этой вершине не были даны ребра, она останется изолированной. Программа поддерживает произвольные псевдографы с любым числом вершин, ребер, в том числе кратных и петлевых.

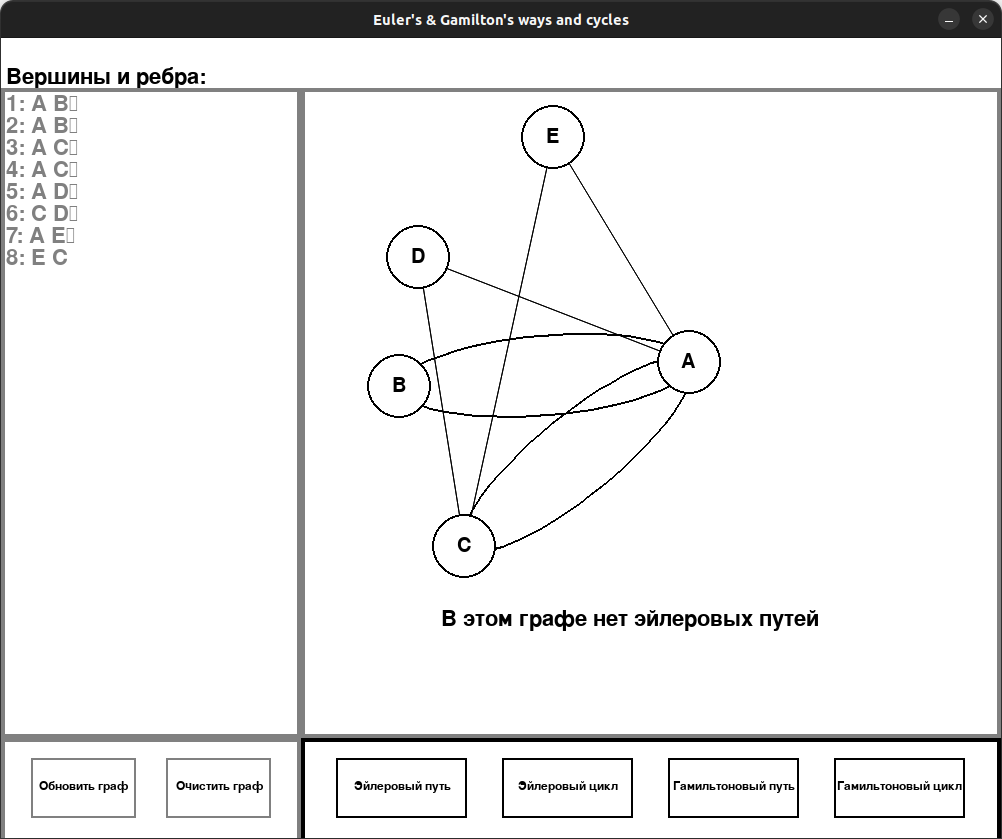
Слева снизу видно окно с кнопками «Update graph» и «Clear graph» - обновить граф и очистить граф соответственно. Первая кнопка обновляет граф согласно введенной информации в окне ввода, вторая очищает (удаляет) граф.

Наибольшее окно — окно отображения графа. В этой области будут отображаться вершины и ребра графа. Вершины графа подвижны и стремятся к стабильному состоянию, их начальное положение задается случайно, а далее силы притяжения и отталкивания придают им конечные положения, некоторые вершины так и остаются гармонически качаться из стороны в сторону.

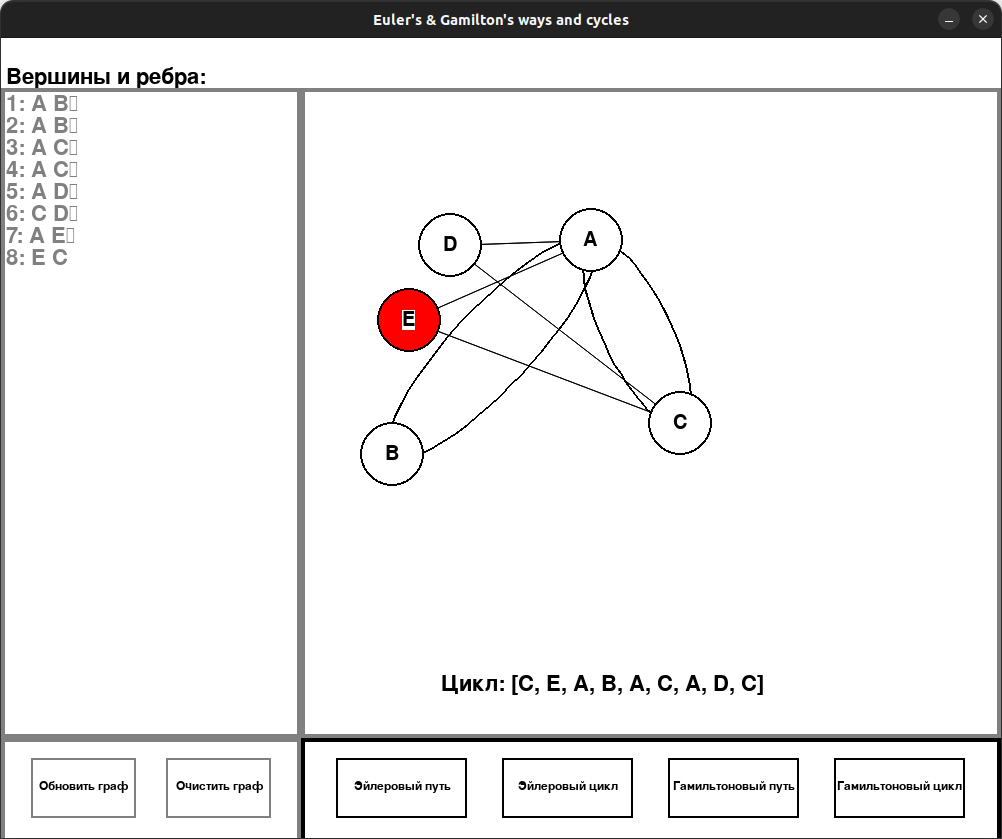
Последнее — нижнее правое окно — окно визуализации обхода соотвествующих путей и цепей. Если для заданного графа существует эйлеровый путь или цикл, он будет выведен текстом в нижней части окна с графом, иначе будет выведено сообщение о том, что таковых пути или цепи не существует. Если заданный граф удовлетворяет Условию Дирака, то программа отображает последовательность вершин, соответствующую пути или циклу.

Дополнительно обход иллюстрирует покраска вершин в красный цвет в порядке пути (цикла).

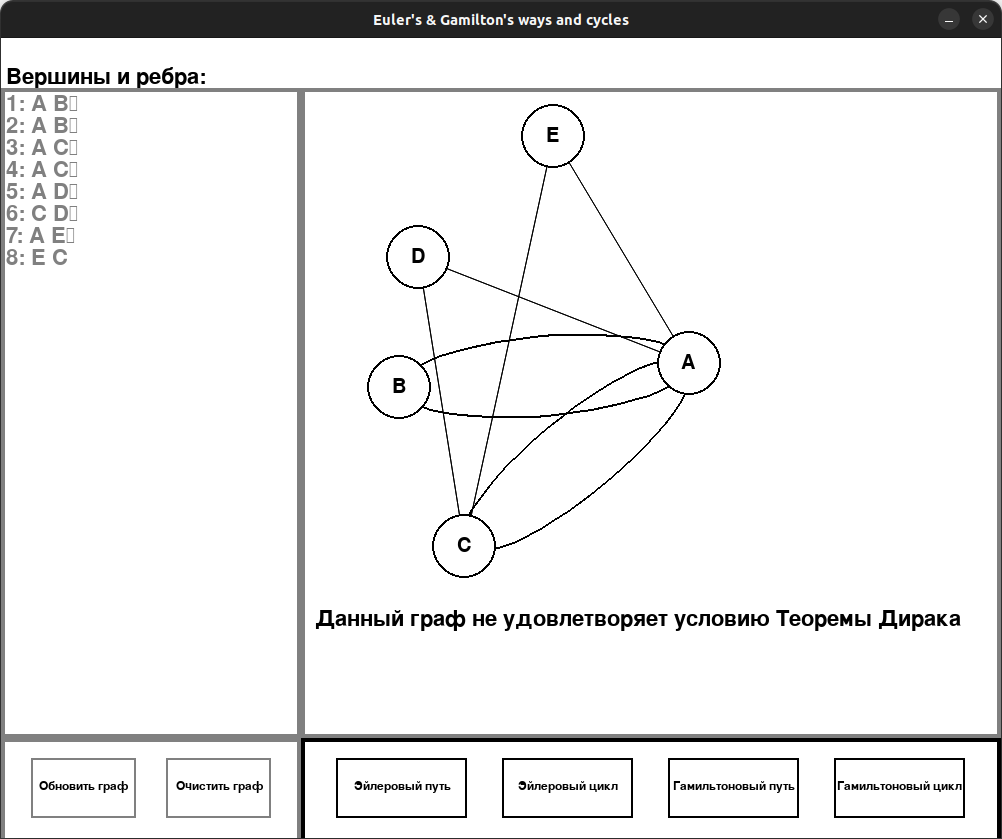
Эйлерового пути нет



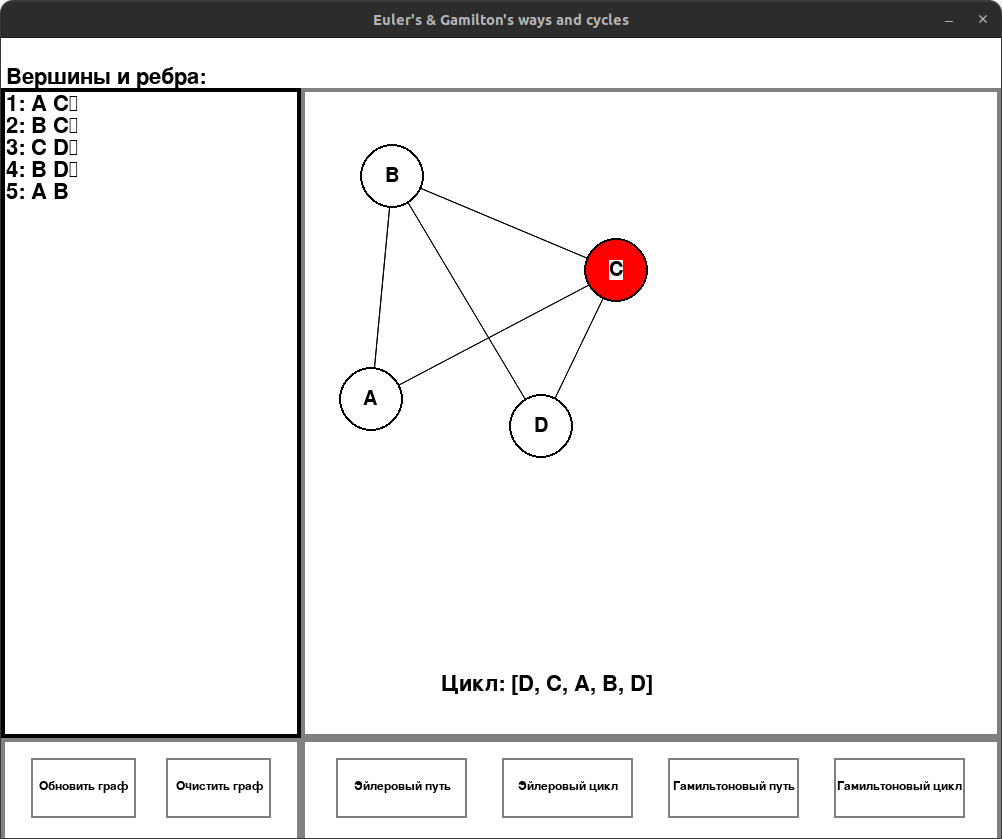
Обход эйлерового цикла



Граф не удовлетворяет Условию Дирака, гамильтоновы пути и циклы не ищутся



Обход гамильтонового цикла



Рассмотрим подробнее последний пример. В графе *G 4* вершины, наименьшая степень вершины *G — 2,* следовательно, по Теореме Дирака найдется гамильтоновый путь. Используем описанный выше алгоритм его нахождения:

*Пусть q := {C,A,D,B}*

*Т.к. (C,A) ∈ E ⇒ q := {A,D,B,C}*

*(A,D) ∉E ⇒*

*i := 2*

*(A,B), (D, C) ∈ E ⇒ i = 2*

*j := 0*

*2 + 0 < 3 - 0 ⇒ q := {A,B,D,C}, j := 1*

*2 + 1 > 3 - 1 ⇒ q = {A,B,D,C}*

*q := {B,D,C,A}*

…

*{B,D,C,A,B}* — гамильтоновый цикл, а это значит, что алгоритм далее будет просто циклически сдвигать последовательность. В конце алгоритм дополнит путь q до цикла, выбрав последней вершиной *D*

**Описание структур данных**

Программа написана на языке Python. Выбор языка программирования обусловлен широкой пользовательской библиотекой и высокой скоростью разработки.

Логически граф представляется как упорядоченная пара двух множеств *G = <V, E>,* где *V —* множество вершин графа, *E —* множество ребер графа. Множества реализуются встроенным классом set. Физически граф представляет класс Graph, полями которого являются:

self.V — множество вершин;

self.E — множество ребер;

self.ortype — тип графа (ориентированный/неориентированный)

self.wtype — тип графа (взвешенный/невзвешенный);

self.delta — наименьшая степень вершин графа;

self.is\_coherent — является-ли граф связным;

self.n — количество вершин;

self.m — количество ребер;

self.scale — масштаб визуализируемого графа;

self.size — визуальный размер вершин;

self.edges — границы для генерации положений вершин,

а методами:

def \_\_init\_\_(self, input, type, wtype, ortype, scale, size, edges) — конструктор класса

def clear(self) — очистить граф;

def copy(self) — копировать граф;

def find\_edge(self, v, u, name, weighted, w) — найти ребро в графе, удовлетворяющее параметрам;

def find\_vertex(self, v) — найти вершину, удовлетворяющую параметрам;

def generate\_positions(self) — функция генерации позиций для вершин;

def copy\_vertex(self) — скопировать вершины графа в отдельное множество;

def remove\_edges(self, E) — вернуть новый граф без указанных ребер;

def remove\_nodes(self, V) — вернуть новый граф без указанных вершин;

def remove\_edges\_update(self, E) — удалить указанные ребра из графа;

def remove\_nodes\_update(self, V) — удалить указанные вершины из графа;

def coherent(self) — проверить, является-ли граф связным;

def update\_positions(self, dt) — обновить позиции вершин, учитывая их нынешние положения, скорости и ускорения за указанный промежуток времени;

def eq\_classes(self) — вернуть разбиение множества ребер на классы эквивалентности по отношению e.vpe.v — e.v инцидентно e.u, где e — ребро, исходящее из e.v и входящее в e.u; (т.е. вернуть множество множеств ответствующих кратных ребер);

def eulers\_way(self) — вернуть эйлеровый путь;

def eulers\_cycle(self) — вернуть эйлеровый цикл;

def least\_degree(self) — найти наименьшую степень вершин графа;

def is\_gamiltons\_graph(self) — проверка связности и Условия Дирака;

def gamils\_cycle(self) — вернуть гамильтоновый цикл;

def gamils\_way(self) — вернуть гамильтоновый путь;

и перегруженные операторы.

Вершины графа представляются классом Vertex, полями которого являются:

self.name — имя вершины;

self.R — визуальный радиус вершины;

self.r, self.v, self.a — координата, скорость и ускорение вершины, выражается классом Point;

self.color — визуальный цвет вершины;

self.index — индекс вершины (по порядку создания);

self.degree — степень вершины

Vertex.count — статическое поле, показывает, сколько всего вершины было создано,

и методы:

def \_\_init\_\_(self, name, R, degree, r, v, a, color) — конструктор класса;

def copy(self) — копия вершины;

def move(self, dt, edges, size) — передвинуть вершину в заданных границах (учитывая, что её размер - size), согласно её координатам, скорости и ускорению на время dt;

def pos(self) — координаты вершины;

и перегруженные операторы.

Ребра графа представляются классом Edge, полями которого являются:

self.name — имя ребра;

self.v, self.u — инцидентные вершины;

self.weighted — взвешенно-ли ребро;

self.w — вес ребра;

self.color — цвет ребра;

self.index — индекс ребра;

Edge.count — статическое поле, показывает, сколько ребер было создано,

и методы:

self.\_\_init\_\_(self, v, u, name, weighted, w, color) — конструктор класса;

self.copy() - копировать ребро, притом поля self.v, self.u указывают на те же вершины, а не на копии;

self.reverse() - вернуть копию развернутого ребра.

Класс Point.

поля:

self.x, self.y — координаты точки,

и методы:

def \_\_init\_\_(self, x, y) — конструктор класса;

def copy(self) — вернуть копию точки;

sef scalar(self, other) — скалярное произведение радиус-векторов на точках в стандартном базисе;

def len(self) — длина радиус-вектора на точке;

def pos(self) — вернуть пару (int(self.x), int(self.y)) — визуальные координаты;

и перегруженные операторы.

**Реализация алгоритмов**

Следующие реализации дословно повторяют описанные выше алгоритмы, уточненные на выбранные структуры данных.

**Поиск эйлерового пути**

def eulers\_way(self, vs=None):

if self.is\_coherent == None and self.m > 0:

self.is\_coherent = self.coherent()

if self.is\_coherent and len([v for v in self.V if v.degree % 2 == 1]) != 2:

return []

else:

eulers\_way = []

\_G = self.copy()

st = list()

if vs is None:

for v in \_G.V:

if v.degree % 2 == 1:

st.append(v)

break

else:

st.append(\_G.find\_vertex(vs))

while len(st) > 0:

v = st[-1]

if v.degree == 0:

eulers\_way.append(v)

st.pop(-1)

else:

for \_e in \_G.E:

if v == \_e.v:

e = \_e

break

st.append(e.u)

\_G.remove\_edges\_update(set([e]))

return eulers\_way

**Поиск эйлерового цикла:**

def eulers\_cycle(self):

if self.is\_coherent == None and self.m > 0:

self.is\_coherent = self.coherent()

if self.is\_coherent and len([v for v in self.V if v.degree % 2 == 1]) > 0:

return []

else:

bridge = next(iter(self.E))

v = bridge.v

\_G = self.remove\_edges(set([bridge]))

eulers\_cycle = \_G.eulers\_way(bridge.u)

eulers\_cycle.append(v)

return eulers\_cycle

**Поиск гамильтонового цикла:**

def gamils\_cycle(self):

if not self.is\_gamiltons\_graph():

return []

q = list(self.V)

for k in range(0, self.n \* (self.n - 1)):

if self.find\_edge(q[0], q[1]) is None:

i = 2

while self.find\_edge(q[0], q[i]) is None or \

self.find\_edge(q[1], q[i + 1]) is None:

i += 1

j = 0

while 1 + j < i - j:

q[1 + j], q[i - j] = q[i - j], q[1 + j]

j += 1

q += [q[0]]

q = q[1:]

return [q[-1]] + q

**Поиск гамильтонового пути:**

def gamils\_way(self):

if not self.is\_gamiltons\_graph():

return []

\_G = self.copy()

v = Vertex('R')

for u in \_G.V:

\_G.E.update([Edge(v, u), Edge(u, v)])

v.degree = \_G.n

\_G.V.add(v)

q = \_G.gamils\_cycle()[1:]

i = 0

while q[i] != v:

i += 1

q = q[i+1:] + q[:i]

return q

# Практическое применение

Помимо образовательных целей, эта программа может найти свое применение при построении туристических и проверочных маршрутов. Поиск эйлеровых путей и циклов может быть полезен, если целью экскурсии или туристического маршрута являются набережные или уникальные улицы какого-нибудь города, а поиск гамильтоновых путей и циклов, например, поможет тем же туристам построить маршрут, проходящий через каждый из интересующий их городов единожды.

Так же эти алгоритмы позволяют пользователю произвести перебор мест при поиске чего-либо. Допустим, менеджер офиса при переходе между кабинетами и этажами утерял свой пропуск. Вместо того, чтобы дожидаться помощи персонала, он может найти его самостоятельно, сделав эффективный перебор мест, в которых он был в последнее время.

# Источники и полезные ссылки:

* [Мой githab с исходным кодом приложения](https://github.com/16yo/graph_visualizer/tree/main),
* [Алгоритм нахождения эйлерова пути и цикла](https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_нахождения_Гамильтонова_цикла_в_условиях_теорем_Дирака_и_Оре) (maximal),
* [Алгоритм нахождения гамильтонова пути и цепи](https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_нахождения_Гамильтонова_цикла_в_условиях_теорем_Дирака_и_Оре) (итмо),
* [Статья Википедии про гамильтонов граф](https://ru.wikipedia.org/wiki/Гамильтонов_граф) (wiki),
* [Алгоритм посторения эйлерова пути и цикла](https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_построения_Эйлерова_цикла) (итмо),
* [Вдохновитель идеи](https://csacademy.com/app/graph_editor/).