Санкт-Петербургский государственный университет

**АПАЛЬКО Данил Фаддеевич**

**Выпускная квалификационная работа**

***Нахождение гамильтонова цикла с использованием методов решения линейных систем с применением модульного сложения***

Уровень образования: бакалавриат

Направление 01.03.02 «Прикладная математика и информатика»

Основная образовательная программа CB.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»

Профиль «Математическое и программное обеспечение вычислительных машин»

Научный руководитель:

профессор, кафедра информационных систем,

д. ф.-м. н. Олемской Игорь Владимирович

Рецензент:

профессор, кафедра прикладной математики,

д. т. н. Голоскоков Дмитрий Петрович

Санкт-Петербург

2020

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc41414905)

[1. ТЕОРИЯ ДЛЯ ОБОСНОВАНИЯ МЕТОДА 4](#_Toc41414906)

[1.1. Основные определения 4](#_Toc41414907)

[1.2. Циклическое пространство графа 8](#_Toc41414908)

[1.3. Пространство разрезов 10](#_Toc41414909)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 14](#_Toc41414910)

[2.1 Описание метода с применением теории 14](#_Toc41414911)

[2.2 Формализация метода 19](#_Toc41414912)

[2.3 Оценка алгоритмической сложности метода 21](#_Toc41414913)

[ЗАКЛЮЧЕНИЕ 24](#_Toc41414914)

[СПИСОК ЛИТЕРАТУРЫ 26](#_Toc41414915)

[ПРИЛОЖЕНИЕ А 28](#_Toc41414916)

[Реализации алгоритма на языке программирования Java, с использованием побитовых операций 28](#_Toc41414917)

ВВЕДЕНИЕ

Задача о поиске и существовании гамильтонова цикла в простых графах является классической NP полной [[4]](#R_M_Karp_in_1972), которая известна своей теоретической и вычислительной сложностью. Несмотря на последние достижения в области теории графов, эта неопределенность представляет собой неизбежную проблему для теории алгоритмов. За последние десятилетия гамильтоновы циклы были достаточно хорошо изучены. А так же, благодаря своей вычислительной сложности, нашли применение в области криптографии и используются в системе так называемых протоколов с [нулевым разглашением](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE_%D1%81_%D0%BD%D1%83%D0%BB%D0%B5%D0%B2%D1%8B%D0%BC_%D1%80%D0%B0%D0%B7%D0%B3%D0%BB%D0%B0%D1%88%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC) для передачи зашифрованной информации. Одним из направлений в исследованиях являются условия их существования. Большинство этих условий основаны на сумме степеней узлов графа. Их применение в методах нахождения гамильтонова цикла обычно требует большого количества ребер. Однако часто, если результаты имеют смысл - существуют контрпримеры среди аналогичного вида графов, когда эти условия ослаблены. Другое направление в решении задачи заключается в разработке случайных алгоритмов, которые обычно находят циклы Гамильтона с высокой вероятностью или успешно справляются с определенными классами графов.

*Целью и задачей* данной работы является теоретическая постановка, обоснование и разработка *метода по нахождению всех гамильтоновых циклов для класса разряженных графов с использованием свойств циклического пространства*, а также практическая реализация алгоритма для наглядной проверки полученных результатов.

1. ТЕОРИЯ ДЛЯ ОБОСНОВАНИЯ МЕТОДА

1.1. Основные определения

Пусть — неориентированный граф определенный на множестве узлов и множестве ребер , где , — соответственно количество узлов и ребер графа .

Дальнейшее изложение будет вестись в предположении, что [граф](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#%D0%B3%D1%80%D0%B0%D1%84) является *простым* — в котором нет [кратных рёбер](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%BE%D1%81%D1%81%D0%B0%D1%80%D0%B8%D0%B9_%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D0%B8_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2#%D0%BA%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B5_%D1%80%D1%91%D0%B1%D1%80%D0%B0) и петель, данное условие не обязательно и накладывается для удобства и лаконической составляющей доказательств.

Итак, для теоретического обоснования нам понадобится ряд определений:

Определение 1.1.

*Маршрут* — последовательность узлов , возможно повторяющихся, графа , где каждой паре ставится в соответствие ребро для всех .

Определенный подобным образом маршрут *проходит* по ребрам и узлам .

Маршрут *замкнут*, если .

Ребра, по которым проходит маршрут, *не обязательно различны*.

На [рисунке 1.1](#Route) изображен маршрут, состоящий из 5 узлов и 5 ребер.



Рисунок .1 - Маршрут 1-2-3-4-5-1

Определение 1.2.

*Путем* называется маршрут не имеющий повторяющихся ребер. *Путь* так же является подграфом графа , и состоит из компонент пути - узлов и ребер.

Узел называется *началом*, а узел — *концом* пути.

Путь называется *простым*, когда все его узлы попарно различны.

*Длина* пути — это общее количество его ребер.

На [рисунке 1.2](#Simple_Path) изображен простой путь, длина которого равна трем.



Рисунок 1. - Простой путь 1-3-4-2

Определение 1.3.

*Циклом* называется последовательность узлов и различных ребер графа , где каждой паре ставится в соответствие ребро для всех , при .

Определенный подобным образом цикл *проходит* по ребрам и узлам .

Аналогично пути *цикл* является подграфом графа , и из узлов и рёбер, его составляющих.

На [рисунке 1.3](#Cycle) изображен цикл, состоящий из 5 узлов и 6 ребер.



Рисунок 1. - Цикл 1-2-3-1-4-5-1

Цикл называется *простым*, когда все его узлы попарно различны.

*Длина* цикла — это общее количество его ребер.

На [рисунке 1.4](#Simple_Cycle) изображен цикл, длина которого равна 5.



Рисунок 1. - Простой цикл 1-2-3-5-4-1

Определение 1.4.

Если — простой цикл, и , — два его несмежных узла.

При том , тогда ребро называется *хордой* цикла .

 На [рисунке 1.5](#Chord) изображен простой цикл, содержащий хорду.



Рисунок 1. - Простой цикл 1-2-3-4-5-6-1, черным выделена содержащаяся в нем хорда 2-6

Определение 1.5.

Простой цикл графа не содержащий хорд называется *порожденным циклом*.

На [рисунке 1.6](#Generated_cycle) изображен пример порожденного цикла в графе.



Рисунок 1. - Граф, черным выделен один из содержащихся в нем порожденных циклов 6-7-8-6

Определение 1.6.

Простой путь, проходящий по всем узлам графа — это *гамильтонов путь*.

На [рисунке 1.7](#Hamilton_Path) изображен пример гамильтонова пути.



Рисунок 1. - Граф, черным выделен гамильтонов путь 2-7-3-4-8-6-1-5

Определение 1.7.

Простой цикл, проходящий по всем узлам графа —это *гамильтонов цикл*. Граф — *гамильтонов*, когда в нем присутствует гамильтонов цикл.

На [рисунке 1.8](#Hamilton_Cycle) изображен пример гамильтонова пути.



Рисунок 1. - Граф, черным выделен гамильтонов цикл 2-1-6-8-5-4-3-7-2

1.2. Циклическое пространство графа

Нам понадобится знание о том, что такое линейное пространство. Мы будем использовать линейные пространства над полем . Для возможности в дальнейшем обобщить теорию на ориентированные графы вместо операции сложения по модулю используем операцию симметрической разности, поскольку они эквивалентны для поля .

Определение 1.8.

*Реберным пространством* графа называется линейное пространство над полем , и обозначается . состоит из подмножеств множества . Сумма элементов определяется как их симметрическая разность .

На [рисунке 1.9](#Sum) изображен пример суммы элементов.



Рисунок 1.9 - Пример суммы элементов

 есть линейное пространство над с базисом . Следовательно, .

Определение 1.9.

*Скалярное произведение* , где , определяется как остаток от деления на .

*Ортогональное дополнение* подпространства реберного пространства задается следующим образом:

Замечание 1.1.

Кроме того, элемент записывается в виде , где множитель и равен единице, когда .

При этом если и , то из скалярное произведение можно представить в виде , что соответствует скалярному произведению в более общепринятом смысле.

Для четного же количества ребер истинно равенство . Из-за чего нельзя прибегать к свойствам скалярного произведения над вещественным полем, так как для их доказательства необходима невырожденность. Вместе с тем, становятся невыполнимы стандартные свойства подпространства реберного пространства такие как и .

Определение 1.10.

*Циклическим пространством* графа называется подпространство , сформированное из реберных множеств простых циклов.

В циклическом же пространстве содержится полезная для данного исследования информация о графе .

Лемма 1.1.

*В графе реберные множества порожденных циклов формируют циклическое пространство* .

Доказательство.

Пространство сформировано множествами ребер простых циклов, поэтому если показать, что для простого цикла его реберное множество сформировано множествами ребер порожденных циклов, то мы получим требуемое. Воспользуемся индукцией по количеству ребер. Пусть — не порожденный цикл. Тогда простой цикл имеет хорду , которая разделяет его на два подцикла и . Из чего следует, что , а для и утверждение доказано. ∎

1.3. Пространство разрезов

Определение 1.11.

*Разрезом* называется множество ребер производящие произвольное разбиение множества узлов на два непересекающихся множества и , обозначается как .

*Пространство разрезов* — это подпространство , сформированное всем множеством разрезов.

Для узлов разрез из всех смежных узлу ребер будет определяться как .

Замечание 1.2.

Разрез состоит только из ребер графа, разделяющих любые две части множества его узлов.

Лемма 1.2.

1. *Если*  *и* — *два разреза, то и*  — *разрез*.

2. *Множество порождено всеми разрезами графа и пустым множеством* .

3. *Разрезы*  *формируют* .

Доказательство.

1. Пусть разрезы и . Тогда — множество, состоящее из тех ребер, что пересекают только одну из пары разрезов, то есть концы ребер располагаются в смежных частях одного из двух разбиений . Поэтому и

— разрез.

2. Следовательно, все множество разрезов замкнуто относительно операции симметрической разности.

3. Пусть — любой из разрезов. Тогда сумма всех разрезов вида , где есть , так как сумма — это операция симметрической разности, из чего следует, что при сложении разрезов все ребра между узлами сократятся и останутся лишь ребра разреза . ∎

Определение 1.12.

Пусть остовное дерево связного графа.

Для каждого из ребер граф содержит две компоненты связности — и . -*разрезами* называются все разрезы вида .

Для каждого из ребер граф содержит только один цикл . -*циклами* называются все циклы типа .

На [рисунке 1.10](#U_Cycle) изображен пример -цикла.



Рисунок 1.10 - Черным выделен U-цикл, полученный добавлением ребра (3, 4)

Теорема 1.1.

*Если — остовное дерево связного графа , то выполняются дальнейшие утверждения:*

1*. -циклы формируют базис циклического пространства и*

|  |  |
| --- | --- |
|  | (1.1) |

2. *-разрезы формируют базис пространства разрезов и*

|  |  |
| --- | --- |
|  | (1.2) |

3. *.*

4.*.*

Доказательство.

1. Каждое из ребер входит в один из -циклов . Следовательно, множество -циклов линейно независимы в . Далее, для любого элемента , пусть — каждое ребро из . Тогда элемент принадлежит , а его ребра есть ребра дерева . Следовательно этот элемент равен , поэтому . Откуда можно сделать вывод, что -циклы формируют базис , а размерность есть .

2. Каждое из ребер входит в один из -разрезов . Следовательно, множество -разрезов линейно независимы в . Пусть любой элемент и — все ребра из . При чем элемент принадлежит и ни одно из его ребер не является ребром дерева . Если , то, как следует из [леммы 1.2](#Lemma_1_2) — — разрез графа . Так как граф связный, то, как минимум одно ребро остовного дерева соединяет c , откуда получаем противоречие. Следовательно, , поэтому . А последнее означает, что -разрезы образуют базис , причем .

3. Пусть . Из третьего пункта [леммы 1.2](#Lemma_1_2) получаем, что лишь тогда, когда для любого узла , а именно то, что степень любого узла графа четна. А по [лемме 1.1](#Lemma_1_1) это означает, что .

4. Каждый цикл пересекает разрез только четное число раз, так как покинув , цикл должен вновь попасть в . Следовательно, .

Пусть , и . Как доказано выше, все -разрезы . Тогда и принадлежит . По построению, . Если , то рассматривая ребро и -цикл . Получаем, что , противоречие. Поэтому , и, следовательно, что . ∎

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Описание метода с применением теории

В общем случае поиск гамильтонова цикла, к примеру и для всегда гамильтоновых, планарных графов с максимальной степенью узлов три, остается NP-полной задачей со сложностью , то есть равен множеству всех перестановок от 1 до n, где n — количество узлов графа . Один из первых точных алгоритмов перебора — алгоритм Мартелло. Для уменьшения сложности до могут быть использованы методы динамического программирования, которые, как видно из оценки сложности алгоритма, также зависят лишь от количества узлов в графе.

Поставим задачу описать такой алгоритм по нахождению цикла Гамильтона, сложность которого варьируется от количества ребер на графе . И для разряженный графов, на которых выполняется неравенство , он имел сложность меньшую чем ; выведем .

Рассмотрим граф , изображенный на [рисунке 2.1](#Graph_A), определим его свойства.



Рисунок 2.1 - Граф

Граф односвязный, неориентированный, не имеет петель или кратных ребер, количество узлов в графе , ребер . Матрица инцидентности графа А изображена на [таблице 1](#Incident_Matrix).

Таблица Матрица инцидентности графа А

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1: | 2: | 3: | 4: | 5: | 6: | 7: | 8: | 9: | 10: | 11: |
| 1: | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2: | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3: | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4: | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5: | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 6: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 7: | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

Итак, по определению на графе задано его циклическое пространство над полем , порождаемое реберным множеством простых циклов, необходимо его найти. Представим матрицу инцидентности как линейную систему над полем , сложение в которой будет задано согласно определению реберного пространства графа, а именно — сумма элементов определяется как поразрядное сложение по модулю . Эта система есть:

|  |  |
| --- | --- |
|  | (2.1) |

Получим решение системы [(2.1)](#Formula_2_1) с помощью метода Гаусса:

|  |  |
| --- | --- |
|  | (2.2) |

При перестановке столбцов

Выразим простые циклы матрицы [(2.2)](#Formula_2_2) через зависимость базисного минора размерности , от независимых переменных, получим -циклы, изображенные на [рисунке 2.2](#Basis_From_Simple_Cycles), следующего вида:

|  |  |
| --- | --- |
|  | (2.3) |

где , — -циклы над полем , в -м столбце соответствует принадлежности соответствующего ребра -циклу, а об отсутствии ребра.



Рисунок 2.2 - Базис, построенный из простых циклов графа А

Количество которых, согласно формуле [(1.1)](#Formula_1_1) в [теореме 1.1](#Theorem_1_1) о размерности базиса циклического пространства:

где — размерность базиса циклического пространства графа ;

 — размерность ребер графа ;

 — размерность вершин графа .

В дополнение к этому, попарно суммируем те базисные элементы из [(2.3)](#Formula_2_3), длины циклов которых при этом уменьшаются, до тех пор, пока не останется таких базисных пар, таким образом получаем множество порожденных циклов, изображенных на [рисунке 2.3](#Basis_From_Generated_Cycles), которые, по [лемме 1.1](#Lemma_1_1), формируют циклическое пространство :

|  |  |
| --- | --- |
|  | (2.4) |

где , — элемент базиса над полем , в -м столбце соответствует принадлежности соответствующего ребра элементу, а об отсутствии ребра в элементе.



Рисунок 2.3 - Базис, построенный из порожденных циклов графа А

Гамильтонов цикл — это простой цикл, поэтому он принадлежит циклическому пространству , следовательно равен одному из подмножества множества . Количество всех подмножеств, лежащих в равно , исключив , так как при этом не будет задействован не один цикл, наконец получаем , при условии, что , так как в противном случае граф не является гамильтоновым. Итак, мы имеем верхнюю оценку количества операций, необходимых для поиска цикла Гамильтона., который может быть найден с помощью применения линейной комбинации базисных элементов циклического пространства.

Для нашего графа A по формуле о числе подмножеств конечного множества получаем количество различных подмножеств , то есть для получения всех гамильтоновых циклов необходимо перебрать циклическое пространство раз.

Перебирая линейные комбинации базисных элементов [(2.4)](#Formula_2_4), получаем, что граф — гамильтонов, а гамильтоновы циклы состоят следующих из сумм:

|  |  |
| --- | --- |
|  | (2.5) |

где , — гамильтонов цикл графа , в -м столбце соответствует принадлежности соответствующего ребра циклу Гамильтона, а об его отсутствии.

Таким образом получаем все гамильтоновы циклы графа [(2.5)](#Formula_2_5), изображенные на [рисунке 2.4](#All_Hamilton_Cycles), за конечное число операций.



Рисунок 2.4 - Гамильтоновы циклы графа A, слева направо:

2.2 Формализация метода

Пусть — неориентированный граф без петель и кратный ребер; — количество его ребер; — количество его вершин.

Алгоритм по нахождению всех гамильтоновых циклов графа состоит в следующем:

1) Проверка условий *разреженности* [(2.6)](#Formula_2_6) на графе и *минимального количества ребер* [(2.7)](#Formula_2_7), а именно:

|  |  |
| --- | --- |
|  | (2.6) |
|  | (2.7) |

Если условия [(2.6)](#Formula_2_6) и [(2.7)](#Formula_2_7) выполняются — перейти к пункту [2](#Alg_Item_2), иначе — выход из алгоритма с ошибкой: “Граф не удовлетворяет условиям разреженности или минимального количества ребер”.

2) Сформировать матрицу инцидентности графа *G*. Перейти к пункту [3](#Alg_Item_3).

3) Применить к матричному уравнению [(2.8)](#Formula_2_8) метод Гаусса над полем по модулю , в получившемся решении переменных — зависимые и переменных — независимые.

|  |  |
| --- | --- |
|  | (2.8) |

где — матрица инцидентности графа размерности ;

 — вектор-столбец, состоящий из — элементов;

 — нулевой вектор-столбец размерности .

Перейти к пункту [4](#Alg_Item_4).

4) В полученной на [3](#Alg_Item_3) шаге матрице ступенчатого вида выразить зависимые переменные через независимые, последовательной подстановкой в вектор независимых переменных или , так чтобы любые два вектора были ортогональны между собой. Векторы полученных зависимостей есть -циклы графа , количество которых согласно формуле [(1.1)](#Formula_1_1):

Например, если и , а и — независимые переменные, то векторы выглядят следующим образом:

Перейти к пункту [5](#Alg_Item_5).

5) Над получившимися -циклами графа попарно применять операцию [суммы](#Def_1_8), до тех пор, пока длинны циклов не перестанут уменьшаться — получим базис циклического пространства состоящего из множества порожденных циклов . Перейти к пункту [6](#Alg_Item_6).

6) Перебрать все линейные комбинации сумм базисных элементов вида:

|  |  |
| --- | --- |
|  | (2.9) |

где — -й цикл из циклического пространства ;

 — попарно ортогональные векторы коэффициентов , коэффициент ;

 — базисные элементы циклического пространства .

Перейти к пункту [7](#Alg_Item_7).

7) Проверить какие из циклов , полученных по формуле [(2.9)](#Formula_2_9), являются гамильтоновыми. Таким образом мы получили все гамильтоновы циклы графа .

Последняя проверка требуется из-за возможного случая, изображенного на [рисунке 2.5](#Cycles_With_2_Component), когда n ребер удовлетворяют нескольким циклам в разных компонентах связности.



Рисунок 2.5 - Циклы общем длины n с двумя компонентами связности

Полный пример кода для одного из языков программирования, алгоритма по нахождению всех гамильтоновых циклов для неориентированного разряженного графа, приведен в [приложении А](#Application_A).

2.3 Оценка алгоритмической сложности метода

Подсчитаем оценку алгоритмической сложности данного метода по нахождению всех гамильтоновых циклов разряженного неориентированного графа.

Во-первых, рассмотрим асимптотическую сложность поиска всех базисных элементов циклического пространства методом Гаусса для системы следующего (матричного) вида:

|  |  |
| --- | --- |
|  |  |

где — матрица инцидентности графа ;

 — вектор-столбец;

 — нулевой вектор-столбец.

Разобьем вычисления по порядку выполнения на прямой ход метода Гаусса и обратный.

Прямой ход метода Гаусса состоит в последующем исключении неизвестных из линейных уравнений системы. На итерации производится исключение неизвестной для всех уравнений с индексом большим чем . Для этого для каждого из этих уравнений производится операция суммы над полем с уравнением , которое представляет из себя вектор в пространстве . Поскольку операция производится с каждый элементом строки в количестве , а обрабатываемых строк , число операций в прямом ходе Метода Гаусса определяется как:

|  |  |
| --- | --- |
|  | (2.10) |

где — количество операций прямого хода метода Гаусса.

Для обратного хода на каждой — итерации производится операций суммирования c элементами:

|  |  |
| --- | --- |
|  | (2.11) |

где — количество операций обратного хода метода Гаусса.

Следовательно общая вычислительная сложность поиска всех базисных элементов циклического пространства составляет:

|  |  |
| --- | --- |
|  | (2.12) |

где — общее количество операций необходимые для поиска базисных элементов;

, — количество операций [(2.10)](#Formula_2_10) и [(2.11)](#Formula_2_11) соответственно.

После раскрытия скобок с использованием формул суммирования, определив член с наибольшей степенью, получаем что асимптотическая сложность равна:

|  |  |
| --- | --- |
|  | (2.13) |

где — асимптотическая сложность поиска всех базисных элементов циклического пространства .

Во-вторых, для получения всех циклов общей длины , необходимо перебрать полученных базисных элементов циклического пространства размерность которого равна , и следовательно вычислительная сложность равна:

|  |  |
| --- | --- |
|  | (2.14) |

где — вычислительная сложность поиска всех циклов общей длины .

Во-третьих, для определения принадлежности n ребер к гамильтонову циклу, так как по асимптотике поиск цикла совпадает с поиском в глубину, необходимо операций:

|  |  |
| --- | --- |
|  | (2.15) |

где — вычислительная сложность определения принадлежности ребер циклу Гамильтона.

Подытожим общую алгоритмическую сложность, через сложности, вычислительные ранее:

|  |  |
| --- | --- |
|  | (2.16) |

где — вычислительная сложность поиска всех гамильтоновых циклов в неориентированном разряженном графе без кратных ребер и петель;

, , — оценки [(2.13)](#Formula_2_13), [(2.14)](#Formula_2_14), [(2.15)](#Formula_2_15) соответственно.

ЗАКЛЮЧЕНИЕ

Среди задач, имеющих экспоненциальную и факториальную сложность решения по времени, крайне важно выделять те классы, которые в зависимости от подхода могут быть решены за более приемлемое время, чем с применением некоторого спектра базовых методов, тем самым снижая общую нагрузку на вычислительную производительность.

В ходе написания курсовой на основе теории свойств циклического пространства и пространства разрезов, доказаны формулы, и зависимости циклического пространства с размерностью ребер и вершин графа определенного класса разряженных графов. С применением алгебраических методов продемонстрирована постановка и дальнейшая разработка метода по нахождению всех гамильтоновых циклов для разряженного неориентированного графа.

Изучена научная литература и основные теоретические аспекты исследования, посредством которых раскрыты ключевые понятия, дополненные поясняющими иллюстрациями; и практическое применение общего класса задач на графах.

Разработан метод, имеющий, кроме абстрактного приложения, применение на практике для матриц, построенных, например, для транспортной сети или сети интернет с каналами передачи данных для каждой из сторон. В частности, если необходимо передать какую-то информацию от сервера до всех пользователей частной сети, таблица маршрутизации которых, обычно, разряжена, а затем вернуть завершающий сигнал на сервер, то, если для такой сети выполняются условия данного метода, можно предварительно применить описанный в данной работе метод, а затем с помощью полученных данных о гамильтоновых циклах настроить маршрутизаторы и получить эффективный способ передачи. Применение находит себя так же в криптографии, где из-за NP сложности задачи нахождения гамильтонова цикла, расшифровывающий ключ кодируется в виде цикла Гамильтона. Далее надлежит сформировывать циклы так, чтобы они удовлетворяли условиям данного метода и расшифровывать закодированную информацию эффективным образом.

Исследована и выведена вычислительная сложность метода, позволяющая перед применением метода получить оценку производительности и сравнить ее с другими методами. Оценка имеет свои особенности, отталкивающиеся от количественного соотношения ребер графа, и дает эффективный результат на определенном классе графов по сравнению с вычислительной сложностью распространенных комбинаторных методов, опирающихся на количество узлов графа.

Описан и протестирован алгоритм на одном из актуальных языков программирования с применением битовых операций, которые посредством ускорения основной операции суммирования векторов и более компактного представления матриц улучшают алгоритмическую производительность алгоритма в целом.

Разработанный метод базируется на доказанной теории поэтому устойчив и не дает сбоев, и доступен для понимания без дополнительного углубления в теорию.

СПИСОК ЛИТЕРАТУРЫ

1) Андерсон, Д.А. Дискретная математика и комбинаторика. / Д.А. Андерсон; ред. С.Н. Тригуб; пер. с англ. М.М. Беловой — М., СПб: ООО “И.Д. Вильямс”, 2004. − 960 с.

2) Ерусалимский, Я. М.: Дискретная математика. Теория и практикум. / Я. М. Ерусалимский — СПб: Лань, 2018. — 476 с.

3) Зыков, А. А. Основы теории графов / А. А. Зыков. — М.: Вуз овская книга, 2004. — 662 c.

4) Karp, R.M. Reducibility among combinatorial problems, Complexity of Computer Computations / R.M. Karp — Plenum Press, 1972. — pp.85-103. DOI: 10.1007/978-3-540-68279-0\_8

5) Кузнецов, О. П. Дискретная математика для инженера / О. П. Кузнецов. — 6-е изд., стер. — СПб.: Лань, 2009. 400 с.

6) Курош, А.Г. Курс высшей алгебры. / А.Г. Курош — 21-е изд., — СПб.: Лань, 2020. — 432 с.

7) Микони, С. В. Дискретная математика для бакалавра: множества, отношения, функции, графы: учебное пособие для вузов / С. В. Микони. — СПб: Лань, 2012. — 186 с.

8) Новиков, Ф.А. Дискретная математика: Учебник для вузов. / Ф.А. Новиков — 2-е изд., — СПб.: Питер, 2013. — 432 с.

9) Новиков, Ф.А. Дискретная математика для программистов. / Ф.А. Новиков — 3-е изд., — СПб.: Питер, 2009. — 384 с.

10) Оре, О. Теория графов. / О. Оре; ред. Н.Н. Воробьeв; пер. с англ. И.Н. Врублевской — 2-е изд., стер. — М.: Наука, 1980. — 336 с.

11) Панюкова, Т.А. Комбинаторика и теория графов / Т.А. Панюкова — 3-е изд. — М.: Ленанд, 2017. — 216 с.

12) Просветов, Г. И. Дискретная математика. Задачи и решения: учебное пособие / Г. И. Просветов. — М.: БИНОМ. Лаборатория знаний, 2008. — 222 с.

13) Седжвик, Р., Уэйн, К. Алгоритмы на Java / Р. Седжвик; К. Уэйн; ред. С.Н. Тригуб; пер. с англ. А.А. Моргунова — 4-е изд. — М., СПб: ООО “И.Д. Вильямс”, 2013. — 848 c.

14) Стенли, Р. Перечислительная комбинаторика. / Р. Стенли; ред. А.М. Вершик; пер. с англ. А.И. Барвинка и А.А. Лодкина. — M.: Мир — 1990. — 440 c.

15) Фаддеев, Д. К. Лекции по алгебре / Д. К. Фаддеев— Изд. 4-е., стереотип. — СПб.: Лань, 2005. - 416 с.

16) Хаггарти, Р. Дискретная математика для программистов. / Р. Хаггарти; ред. С. А. Кулешова; пер. с англ. — М.: Техносфера, 2003. — 320 с.

17) Харари, Ф. Теория графов / Ф. Харари; ред. Г. П. Гаврилов; пер. с англ. В. П. Козырева — 5-е изд. — М.: Ленанд, 2018. — 297 с.

18) Яблонский, С.В. Введение в дискретную математику: учебное пособие для вузов / С. В. Яблонский — 5-е изд. — М.: Высшая школа, 2008. — 384 с.

19) Карпов, В.Д. Теория графов [Электронный ресурс] / В.Д. Карпов —Электрон. текстовые дан. — СПб.: [б.и.], 2009. — Режим доступа: https://logic.pdmi.ras.ru/~dvk/graphs\_dk.pdf, свободный.

20) Хитров, Г. М. Теория графов. ГрафоMann [Электронный ресурс] / Г. М. Хитров — Электрон. текстовые дан. — СПб.: [б.и.], 2013. — Режим доступа: http://www.apmath.spbu.ru/grafomann/, свободный.

ПРИЛОЖЕНИЕ А

Реализации алгоритма на языке программирования Java, с использованием побитовых операций

Приведенный алгоритм печатает все гамильтоновы циклы графа, либо дополнительную информацию об их отсутствии или не выполнении начальных условий:

**import** **java.util.Scanner**;

**import** **java.io.File**;

**import** **java.io.FileNotFoundException**;

**import** **java.util.List**;

**import** **java.util.ArrayList**;

**public** **class** **Hamilton** {

 **private** **int** N;

 **private** **final** **int** M;

 **private** **int**[] Pi;

 **private** **long**[] cycles;

 **private** **long**[] rows;

 **static** **int**[][] matrix;

 **public** **Hamilton**(**long**[] rows, **int** N, **int** M)

 {

 **if** (M < N) **throw** **new** RuntimeException("Граф не

 гамильтонов. Ребер

 меньше чем вершин");

 **if** (M > **2**\*N) **throw** **new** RuntimeException("Эффективная

 работа алгоритма

 обеспечивается при выполнении

 равенства M < 2\*N");

 **this**.N = N;

 **this**.M = M;

 **this**.rows = rows;

 **this**.Pi = **new** **int**[M];

 **for** (**int** i = **0**; i < M; i++) Pi[i] = i;

 System.out.println("N вершин = " + N);

 System.out.println("M ребер = " + M);

 cycles = rows.clone();

 gauss(cycles);

 System.out.println("Матрица Гаусса:");

 **for**(**int** i = **0**; i < N-**1**; i++)

 print(cycles[i], M);

 System.out.println();

 System.out.println("Перестановки ребер Pi[i]:");

 **for**(**int** i = **0**; i < **this**.M; i++)

 System.out.print(Pi[i]+**1** + " ");

 cycles = fundamentalCycles(cycles);

 **this**.N = N;

 System.out.println("\n\nФундаментальные циклы(после

 перестановки):");

 **for**(**int** i = **0**; i < cycles.length; i++)

 {

 print(cycles[i], **this**.M);

 }

 }

 **public** **void** **printHamiltonCycles**() {

 **long** k = **1L**;

 **long** copy;

 **long** cycle;

 **int** count = **0**;

 **while** (**true**)

 {

 copy = k;

 cycle = **0L**;

 **for** (; copy != **0L**; copy ^= Long.lowestOneBit(copy))

 {

 **int** i = Long.numberOfTrailingZeros(copy);

 **if** (i > M)

 **throw** **new** RuntimeException("Неправильно

 построены

 фундаментальные циклы");

 **if** (i >= cycles.length)

 {

 **if** (count == **0**)

 **throw** **new** RuntimeException(

 "Гамильтонова цикла

 не существует.");

 **return**;

 }

 cycle ^= cycles[i];

 }

 **if** (isCycle(rows, cycle, N))

 {

 count++;

 System.out.println("\n########");

 System.out.println("Гамильтонов цикл " + "№" +

 count + ":");

 print(cycle,M);

 System.out.println("Матрица инцидентности:");

 printEdges(rows, cycle);

 System.out.println("Базисная комбинация:");

 print(k, M-N+**1**);

 System.out.println("########");

 }

 k++;

 }

 }

 **private** **boolean** **isCycle**(**long**[] rows, **long** cycle, **int** N)

 {

 **if** (Long.bitCount(cycle) != N) **return**(**false**);

 **long**[] copy = rows.clone();

 **int** sumEdges = **1**;

 **int** current = **0**;

 **long** bit;

 **while** (copy[current] != **0**)

 {

 bit = Long.lowestOneBit(copy[current]);

 copy[current] ^= bit;

 **if** ((cycle & bit) != **0**)

 {

 **for** (**int** j = **0**; j < copy.length; j++)

 {

 **if** (j == current) **continue**;

 **if** ((copy[j] & bit) != **0**)

 {

 copy[current] = **0**;

 copy[j] ^= bit;

 sumEdges++;

 current = j;

 **break**;

 }

 }

 }

 }

 **return** sumEdges == N;

 }

 **private** **void** **gauss**(**long**[] rows)

 {

 **for** (**int** i = **0**; i < N-**1**; i++)

 {

 **long** col = (**1L**<<i);

 **if** ((rows[i] & col) == **0**)

 **if** (!rearrange(rows, i, Pi))

 **throw** **new** RuntimeException("Граф

 не гамильтонов.

 Метод Гаусса: элемент

 для перестановки не найден.");

 **for** (**int** j = **0**; j < N; j++)

 {

 **if** (i == j) **continue**;

 **if** ((rows[j] & col) != **0**)

 rows[j] ^= rows[i];

 }

 }

 **if** (rows[--N] != **0L**)

 **throw** **new** RuntimeException("Граф не гамильтонов.

 Система решений не вырождена");

 **for** (**int** i = **0**; i < N; i++)

 **if** (Long.bitCount(rows[i]) == **1** ||

 Long.bitCount(rows[i]) == **0**)

 **throw** **new** RuntimeException(

 "Граф не гамильтонов.

 Базис не может быть

 построен");

 }

 **private** **long**[] **fundamentalCycles**(**long**[] rows)

 {

 **long**[] cycles = **new** **long**[M-N];

 **long** basis = **1L**<<N;

 **for** (**int** j = **0**; j < M-N; j++)

 {

 cycles[j] = basis;

 **for** (**int** i = **0**; i < N; i++)

 {

 **if** ((rows[i] & basis) != **0**)

 cycles[j] |= **1L**<<i;

 }

 cycles[j] = permutePi(cycles[j]);

 basis <<= **1**;

 }

 **for** (**int** j = **0**; j < M-N; j++)

 {

 **for** (**int** i = j + **1**; i < M-N; i++)

 {

 **if** ((cycles[i] & cycles[j]) != **0** &&

 ((Long.bitCount(cycles[i] ^ cycles[j]) <

 Math.max(Long.bitCount(cycles[i]),

 Long.bitCount(cycles[j])))))

 {

 **if** (Long.bitCount(cycles[i]) <

 Long.bitCount(cycles[j]))

 {

 cycles[j] = cycles[i] ^ cycles[j];

 }

 **else**{

 cycles[i] = cycles[i] ^ cycles[j];

 }

 }

 }

 }

 **return** cycles;

 }

 **private** **boolean** **rearrange**(**long**[] rows, **int** i, **int**[] Pi) {

 **int** tmp = Pi[i];

 **int** changed = i;

 **while** ((rows[i] & (**1L**<<changed)) == **0**)

 **if** (++changed == **this**.M) **return** **false**;

 Pi[i] = Pi[changed];

 Pi[changed] = tmp;

 **long** mask = (**1L**<<i) | (**1L**<<changed);

 **for** (**int** j = **0**; j < rows.length; j++) {

 **long** jBit = rows[j] & (**1L**<<i);

 **long** chgBit = rows[j] & (**1L**<<changed);

 **if** ((jBit != **0** || chgBit != **0**) &&

 !(jBit != **0** && chgBit != **0**)) {

 rows[j] ^= mask;

 }

 }

 **return** **true**;

 }

 **private** **long** **permutePi**(**long** hamilton) {

 **long** back = **0L**;

 **for** (**int** i = **0**; i < M; i++)

 {

 **if** ((hamilton & (**1L**<<i)) != **0**)

 {

 back |= **1L**<<Pi[i];

 }

 }

 **return** back;

 }

 **private** **static** **void** **printEdges**(**long**[] rows, **long** hamilton)

 {

 **for** (**int** i = **0**; i < matrix.length; i++)

 {

 **for** (**int** j = **0**; j < matrix[**0**].length; j++)

 {

 **if** ((hamilton & (**1L**<<j)) != **0**)

 System.out.print(matrix[i][j]);

 }

 System.out.println();

 }

 }

 **private** **static** **void** **print**(**long** row, **int** M)

 {

 String str = "";

 **for** (**int** k = **0**; k < M; k++)

 {

 str += (row & (**1L**<<k)) != **0L** ? "1" : "0";

 }

 System.out.println(str);

 }

 **public** **static** **void** **main**(String[] args)

 {

 List<String> list = **new** ArrayList<>();

 Scanner in;

 **try** {

 in = **new** Scanner(**new** File(args[**0**]));

 }

 **catch**(FileNotFoundException e) {

 System.out.println(e.getMessage());

 **return**;

 }

 **while** (in.hasNextLine()) list.add(in.nextLine());

 String[] lines = list.toArray(**new** String[**0**]);

 **int** N = lines.length;

 **int** M = lines[**0**].length();

 **long**[] rows = **new** **long**[N];

 matrix = **new** **int**[N][M];

 **for** (**int** i = **0**; i < N; i++) {

 **for** (**int** j = **0**; j < M; j++)

 **if** (lines[i].charAt(j) == '1') {

 rows[i] |= **1L**<<j;

 matrix[i][j] = **1**;

 }

 **else** matrix[i][j] = **0**;

 }

 **try**

 {

 Hamilton H = **new** Hamilton(rows, N, M);

 H.printHamiltonCycles();

 }

 **catch**(Exception e){ System.out.println(e.getMessage());

 }

}

Подгружаемый файл должен содержать матрицу инцидентности графа, аналогичного следующему: