Московский авиационный институт

(Национальный исследовательский университет)

институт “Информационные технологии и прикладная математика”

кафедра “Математическая кибернетика”

КУРСОВАЯ РАБОТА

*по курсу* «Дискретная математика»

*2-й семестр*

|  |  |
| --- | --- |
| Тема: | Теория графов, алгебраические структуры, теория алгоритмов.  Нахождение правильной вершинной раскраски графа на основе стратегии перебора. |

|  |  |
| --- | --- |
| *Студент*: | Филиппов Владимир Михайлович |
| *Группа*: | М8О-110Б |
| *Руководитель*: | Н.С. Алексеев |
| *Оценка*: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *Дата*: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

### 

### Введение

В настоящем отчете по курсовой работе приведены результаты, полученные в рамках изучения курса “Дискретная математика” во втором учебном семестре.

В части I приведены решения типовых задач по теории графов, а также представлена реализация алгоритма нахождения правильной вершинной раскраски графа на основе стратегии перебора. Текст программы и тестовые примеры вынесены в Приложение.

В части II приведены решения типовых задач по темам “Алгебраические структуры” и “Теория алгоритмов”.

### Часть I. Теория графов

*Здесь должны содержаться постановки и решения задач А, Б, 1-7.*

### Часть II. Теория алгоритмов и алгебраические структуры

*Здесь должны содержаться постановки и решения задач 7-10.*

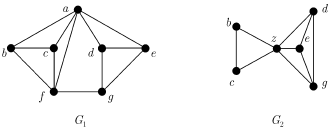
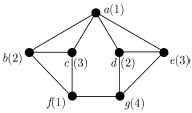
### 

### Часть III. Программная реализация алгоритма нахождения правильной вершинной раскраски графа на основе стратегии перебора.

В основе алгоритма лежит идея о том, что изначальный граф, для которого необходимо найти раскраску можно заменить другими, основываясь на формуле, где *χ(G)* - хроматическое число, для которых она также будет верна. Таким образом, мы приходим к дереву вариантов, которое можно построить при помощи рекурсии.

#### Описание алгоритма.

#### Пусть нам дан произвольный граф *G* (рис 1.1). Выберем в данном графе *G* две несмежные вершины *x* и *y* и построим два новых графа: *G1,* получающийся добавлением ребра *(x,y)* к графу *G, и G2,* получающийся из *G* слиянием вершин *x* и *y.* Операция слияния состоит в удалении вершин *x* иy и добавлении новой вершины *z* и ребер*,* соединяющих ее с каждой вершиной, с которой была смежна хотя бы одна из вершин *x, y.* На рис. 1.2 показаны графы *G1 и G2*, получающиеся из графа *G,* изображенного на рис.1.1*,* с помощью этих операций, если в качестве *x* и *y* взять вершины *a* и *f*.



#### Рис. 10.1 Рис 1.2

Если в правильной раскраске графа *G* вершины *x* и *y* имеют разные цвета, то она будет правильной и для графа *G1*. Если же цвета вершин *x* и *y* в раскраске графа *G* одинаковы, то граф *G2* можно раскрасить в то же число цветов: новая вершина *z* окрашивается в тот цвет, в который окрашены вершины *x* и *y*, а все остальные вершины сохраняют те цвета, которые они имели в графе *G*. И наоборот, раскраска каждого из графов *G1*, *G2*, очевидно, дает раскраску графа *G* в то же число цветов. Поэтому , что дает возможность рекурсивного нахождения раскраски графа в минимальное число цветов. Заметим, что граф *G1* имеет столько же вершин, сколько исходный граф, но у него больше ребер. Поэтому рекурсия в конечном счете приводит к полным графам, для которых задача о раскраске решается тривиально.

#### Описание реализации алгоритма

Мною алгоритм был реализован на языке программирования Python. Для удобной работы с графами была использована библиотека networkx. В файле contraction.py реализованы две функции Painting и IsComplete (см приложение 1). Вторая проверяет является ли граф полным. Подробнее рассмотрим устройство первой. Вначале мы проверяем, является ли наш граф полным и если да, то раскрашиваем все его вершины в разные цвета. После мы дважды копируем изначальный граф. Далее в цикле находим две несвязные вершины. После чего в графе *g1* добавляем ребро, соединяющее эти две вершины, а в графе *g2* склеиваем их. Далее вызываем функцию Painting, только уже от этих графов. Когда алгоритм дойдет до нижнего уровня рекурсии (до полных графов), запустится проверка на минимальное хроматическое число и выберется из двух графов тот, у которого оно наименьшее, после чего граф с верхнего уровня рекурсии раскрасится согласно алгоритму. Таким образом раскрытие рекурсии приведет к раскраске изначального графа.

#### Описание программы и инструкции по работе с ней

В файле main.py происходит работа с файлом. Там мы читаем его и записываем получившуюся раскраску.

Установить программу можно с моего GitHub репозитория при помощи команды:

* *git clone* [*https://github.com/zloyaloha/contraction-graph-coloring-algorithm.git*](https://github.com/zloyaloha/contraction-graph-coloring-algorithm.git)

Для корректной работы необходимо также установить пакеты из файла requirements.txt при помощи команды:

* *pip install -r requirements.txt*

Далее при помощи приложения ГРАФОИД (инструкция к нему также лежит внутри репозитория) необходимо построить граф. В качестве исполняемого файла необходимо выбрать:

* *~\contraction-graph-coloring-algorithm\main\main.exe*

#### Оценка вычислительной сложности алгоритма

#### Время работы удовлетворяет такому же рекуррентному соотношению, как и числа Фибоначчи, поэтому в наихудшем случае алгоритм будет работать за время для n вершин и m ребер.

#### Результаты тестирования программы

Результаты тестирования программы вынесены в Приложение 2. Алгоритм работает корректно для любых графов. Исключением может быть граф, хроматическое число которого превышает 14, так как в программе заложено лишь 14 цветов.

#### Пример прикладной задачи

Существуют многочисленные практические приложения раскраски графов. Когда приложение моделируется как проблема с раскраской вершин графа, то вершины в каждом цветовом классе обычно представляют отдельные субъекты, которые не конкурируют или не конфликтуют друг с другом. Пять основных классов приложений, решаемых с помощью раскраски вершин (1—5) графов, следующие:

1) распределение радиочастот;

2) хранение химических веществ;

3) составление расписаний;

4) распределение регистров в микропроцессорах;

5) политическая картография;

### Заключение

В ходе выполнения курсовой работы были решены 12 задач по различным разделам курса “Дискретная математика”.

Кроме того был изучен вопрос об алгоритмах вершинной раскраски графа. Была написана и отлажена программа, реализующая нахождение правильной вершинной раскраски графа на основе стратегии перебора. Программа написана на языке программирования Python. Программа обеспечивает связь по установленному формату с системой ГРАФОИД, разработанной на кафедре 805, что дает возможность обеспечить графический интерфейс при ее использовании. Эта программа является основным результатом курсового проектирования.

## 

## **Список использованных источников**

* 1. *Раскраска графа // Wikipedia URL: clck.ru/34QrWm (дата обращения: 14.05.2023).*
* 2. *Переборный алгоритм для раскраски // НОУ ИНТУИТ URL: https://intuit.ru/studies/courses/101/101/lecture/2960?page=2 (дата обращения: 14.05.2023).*

### Приложение 1. Текст программы

#файл main.py

* import networkx as nx
* import numpy as np
* from sys import argv
* from contraction import Painting
* file = argv[1]
* colorsList = ["green", "red", "blue", "cyan", "pink", "yellow", "white", "black", "dark-green", "dark-red ", "dark-blue", "dark-cyan", "dark-pink", "dark-yellow"]
* with open(file, "r+") as f:
* size = int(f.readline())
* matrix = []
* for i in range(size):
* line = f.readline().split()
* line = [int(i) for i in line]
* matrix.append(line)
* numpyMatrix = np.array(matrix)
* graphy = nx.from\_numpy\_array(numpyMatrix)
* nx.set\_node\_attributes(graphy, 0, "color")
* res = Painting(graphy)
* f.write("<Vertex\_Colors>\n")
* for line in res[1].nodes(data = 'color', default=1):
* copyLine = " ".join((str(line[0]), colorsList[int(line[1])],))

        f.write(copyLine + "\n")

* f.write("<Text>\nChromatic number of graph is X(G) = " + str(res[0]))

#файл contraction.py

import networkx as nx

import matplotlib.pyplot as plt

def IsComplete(graph):

    n = graph.order()

    s = graph.size()

    return n \* (n - 1) // 2 == s

def Convert(list):

    return {list[i][0]: (list[i][1], list[i][0])  for i in range(len(list))}

def Painting(graph):

    ord = graph.order()

    if (IsComplete(graph)):

        for i in graph.nodes():

            graph.nodes()[i]['color'] = max([i[1] for i in graph.nodes(data = 'color', default=1)]) + 1

        return ord, graph

    g1 = graph.copy()

    g2 = graph.copy()

    nodes = list(graph.nodes())

    for n in nodes:

        nodesNonNeigh = list(nx.non\_neighbors(graph, n))

        if (len(nodesNonNeigh) != 0):

            nodeNonNeigh = nodesNonNeigh[0]

            node = n

            break

    g1.add\_edge(node, nodeNonNeigh)

    nodeAllNeigh = list(nx.all\_neighbors(graph, node))

    nodeNonNeighAllNeigh = list(nx.all\_neighbors(graph, nodeNonNeigh))

    g2.remove\_nodes\_from((node, nodeNonNeigh))

    maxNum = max(graph.nodes())

    g2.add\_node(maxNum + 1, color = graph.nodes()[node]['color'])

    edgesG2 = [(maxNum + 1, N) for N in list(set(nodeAllNeigh) | set(nodeNonNeighAllNeigh))]

    g2.add\_edges\_from(edgesG2)

    xG2 = Painting(g2)

    xG1 = Painting(g1)

    if xG1[0] < xG2[0]:

        for i in xG1[1].nodes():

            graph.nodes()[i]['color'] = xG1[1].nodes()[i]['color']

        nx.draw(graph, labels=Convert(list(graph.nodes(data = 'color', default = 1))), font\_weight='bold')

        plt.suptitle("xg1")

        plt.show()

        return xG1[0], graph

    else:

        for i in xG2[1].nodes():

            if i == max(xG2[1].nodes()):

                graph.nodes()[node]['color'] = xG2[1].nodes()[i]['color']

                graph.nodes()[nodeNonNeigh]['color'] = xG2[1].nodes()[i]['color']

            else:

                graph.nodes()[i]['color'] = xG2[1].nodes()[i]['color']

        return xG2[0], graph

### Приложение 2. Результаты тестирования программы

