Пензенский государственный университет

Кафедра “Вычислительная техника”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовой работе

По курсу “Логика и основы алгоритмизации в инженерных задачах”

На тему “Разработка алгоритма нахождения хроматического числа графа”

Выполнил студент группы 22ВВВ3

Кулахметов С.И.

Приняли:

к.э.н. доцент Акифьев И.В.

к.т.н. доцент Юрова О.В.

Пенза 2023

**Содержание**

Реферат…………………………………………………………………………....5

Введение………………………………………………………………………..…6

Постановка задачи……………………………………………………………..…7

Теоретическая часть задания………………………………………………….…8

Описание алгоритма программы……………………………………………..…9

Описание программы…………………………………………………………...10

Тестирование………………………………………………………………….....17

Ручной расчёт……………………………………………………………………23

Заключение……………………………………………………………………….24

Список литературы………………………………………………………………25

Приложение А. Листинг программы…………………………………………...26

**Реферат**

Отчёт 38 стр., 32 рисунка, 1 таблица, 1 приложение.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ РАСКРАСКИ ГРАФА, ХРОМАТИЧЕСКОЕ ЧИСЛО.

Цель исследования – разработка программы, способной произвести раскраску графа с наиболее оптимизированной затратой цветов, и при помощи данного действия найти хроматическое число заданного графа.

В работе рассмотрены основные правила раскрашивания графа, при помощи которых находится хроматическое число данного графа

**Введение**

Хроматическое число графа - это минимальное число цветов, которыми можно покрасить граф. Для нахождения данного числа можно прибегнуть к простому способу: покрасить граф и посчитать количество цветов, которые нам потребовались.

При раскрашивании графа инициализируется специальный массив, который отвечает за хранение цветов, где каждый индекс - это раскрашиваемая вершина. Перебрав данный вектор в цикле и найдя максимальное число цвета его можно принять за хроматическое.

В качестве сред разработки мною были выбраны среды Code::Blocks IDE и PyCharm Community Edition, языки программирования – Си и Python 3, соответственно.

Целью данной курсовой работы является разработка программы на языках Си и Python 3, которые является широко используемыми. С помощью языка Си в данном курсовом проекте реализуется алгоритм раскраски графа, при помощи которого далее находится его хроматическое число. На языке Python 3 реализована графическая составляющая для визуализации работы алгоритма.

**Постановка задачи**

Требуется разработать программу, которая найдёт хроматическое число графа, раскрасив его, и визуализирует полученный результат.

Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь выбрал необходимую для него функции: “Сгенерировать граф” - для генерирования матрицы смежности графа псевдорандомом и “Считать граф из файла” - для применения алгоритма к графу, который вводит пользователь собственноручно. После обработки сгенерированного или прочтённого графа на экран должен выводиться граф в раскрашенном виде и его хроматическое число. Необходимо предусмотреть различные ошибки при записи пользователем графа в файл, чтобы предотвратить некорректный результат работы программы. Устройства ввода - клавиатура и мышь.

.

**Теоретическая часть задания**

Граф G, изображенный на рисунке 1, состоит из множества вершин X1,X2,…,Xn и множества ребер, соединяющих определенные вершины.

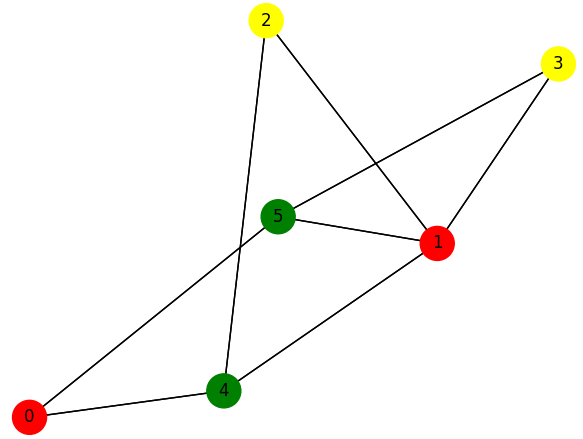


Рисунок 1 – Пример графа

При представлении графа в виде матрицы смежности можно хранить информацию о ребрах между вершинами графа в квадратной матрице, где присутствие пути из одной вершины в другую обозначается «1», а отсутствие пути – «0».

Для нахождения хроматического числа графа одним из наиболее оптимальных является алгоритм «жадной раскраски». Он проходит по каждой вершине графа и выбирает минимальный доступный цвет, который еще не использовался среди соседних вершин. Полученные цвета записываются в специально созданный одномерный массив цветов, при том каждому цвету соответствует своё число. Благодаря особенности работы алгоритма, максимальное число цвета - это по совместительству и общее число цветов, использованных при раскраске графа.

**Описание алгоритма программы**

Для реализации алгоритма нахождения хроматического числа графа необходима матрица смежности графа, промежуточный вектор цветов, выходной вектор цветов и информация о количестве вершин в графе. Цель алгоритма, для начала, заключается в раскрашивании графа. Для этого в цикле мы проходим по каждой вершине и проверяем цвета смежных вершин, после чего исходной вершине присваивается минимальный доступный цвет, который записывается в вектор цветов. Наконец осуществляется перебор вектора цветов с целью найти максимальный, который и является хроматическим числом окрашенного графа.

Ниже представлен псевдокод алгоритма, реализованный в функциях RandGraph(), ReadFromFile() и ColorsInit().

**RandGraph(int size) и ReadFromFile()**

1. для i = 0 пока i < число вершин графа

2. обнулить массив used\_colors

3. для j = 0 пока j < число вершин графа

4. если i и j != 0 и цвет j-вершины != 0

5. использовать j как индекс и установить used\_colors[цвет j-вершины] = 1

6. конец цикла

7. для j = 1 пока j <= число вершин графа

8. если used\_colors[j] == 0

9. colors[i] = j

10. выйти из цикла

11. конец условия

12. конец условия

**ColorsInit()**

число = 0

1. для i = 0 пока i < число вершин графа

2. если «число» < colors[i]

3. «число» = colors[i]

4. конец условия

5. вернуть значение «число»

**Описание программы**

Для написания данной программы были использованы языки программирования Си и Python 3.

Проект был создан в виде графического приложения на базе фреймворка Tkinter.

Данная программа является многомодульной, так как состоит из нескольких файлов и собственных динамических библиотек: RandGraph.so, ReadFromFile.so, main.py, Graphs.py и Colors.py.

Работа программы начинается с вывода информации о курсовой работе, а также с предложения пользователю «Сгенерировать граф», «Считать граф из файла» или «Завершить» работу, если пользователь передумал (рис. 2).

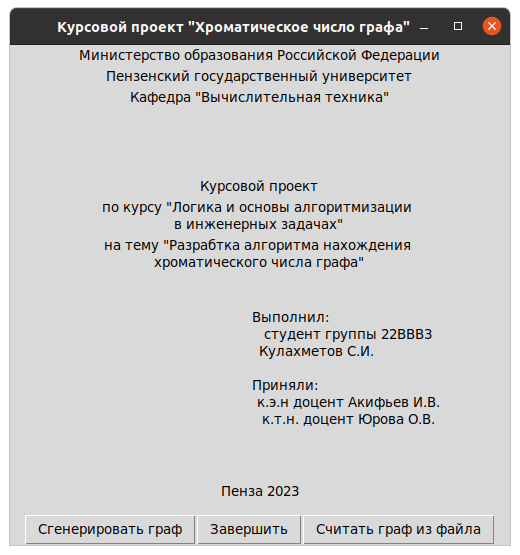


Рисунок 2 - Начало работы программы

При выборе варианта «Сгенерировать граф», запускается функция RandomGraph(), которая в свою очередь вызывает функцию RandGraph(int size), но уже из динамической библиотеки RandGraph.so. В ней, в свою очередь, выполняется псевдорандомное генерирование матрицы смежности (рис. 3) и раскрашивание графа (рис. 4). При запуске открывается поле ввода, в котором нам необходимо ввести количество вершин генерируемого графа (рис. 5). Число вершин должно быть от 1 до 20. Как только мы введём значение (рис. 6), сработает скрипт на языке Python 3, который при помощи модулей matplotlib и networkx отрисует раскрашенный ранее в функции RandGraph() граф и выведет найденное в функции ColorsInit() хроматическое число (рис. 7).

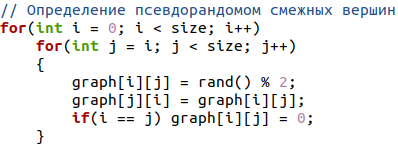


Рисунок 3 - Генерирование матрицы смежности при помощи функции rand()

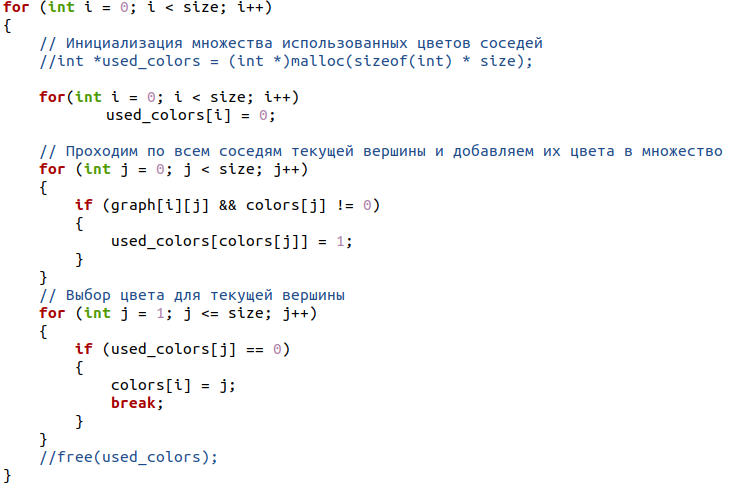


Рисунок 4 - Раскрашивание графа

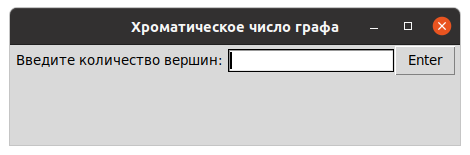


Рисунок 5 - Поле для ввода количества вершин графа

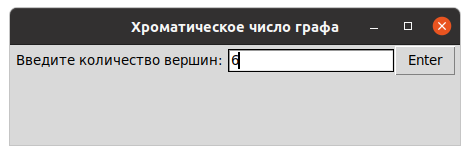


Рисунок 6 - Заполненное поле ввода

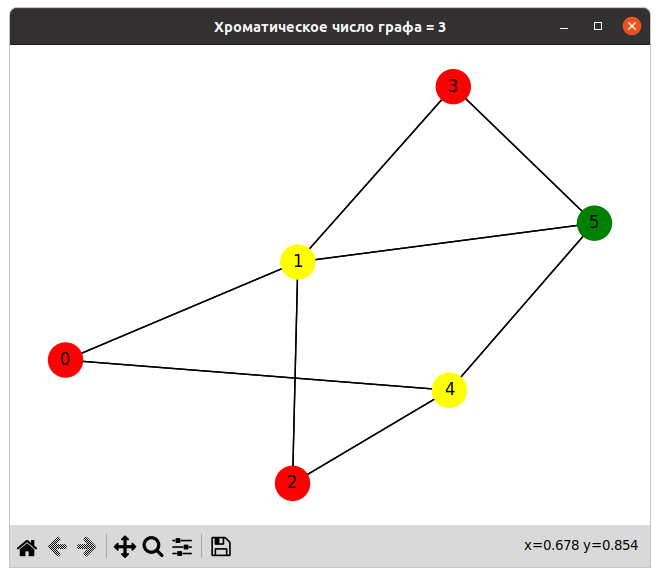


Рисунок 7 - Отрисовка раскрашенного графа с его хроматическим числом

В случае, если пользователь выбирает «Считать граф из файла», открывается аналогичное поле ввода, однако в нём уже требуется ввести имя файла с расширением через точку (рис. 8), в котором записана матрица смежности. Матрица смежности должна иметь вид, аналогичный представленному на рисунке 9. Граф считывается (рис. 10) функцией ReadFromFile() из динамической библиотеки ReadFromFile.so и обрабатывается тем же алгоритмом, что представлен на рисунке 4. После чего аналогично примеру с генерацией выводится в раскрашенном виде с указанием хроматического числа (рис. 11)

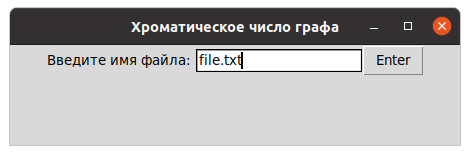


Рисунок 8 - Ввод имени файла с расширением

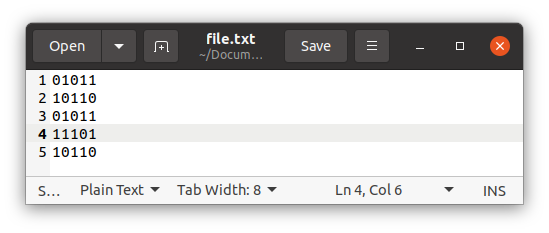


Рисунок 9 - Матрица смежности, записанная в файле

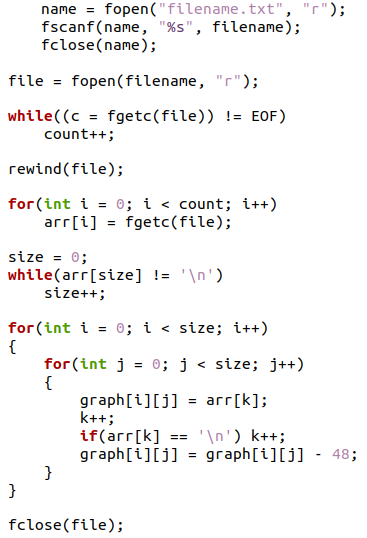


Рисунок 10 - Считывание данных из указанного файла на языке Си

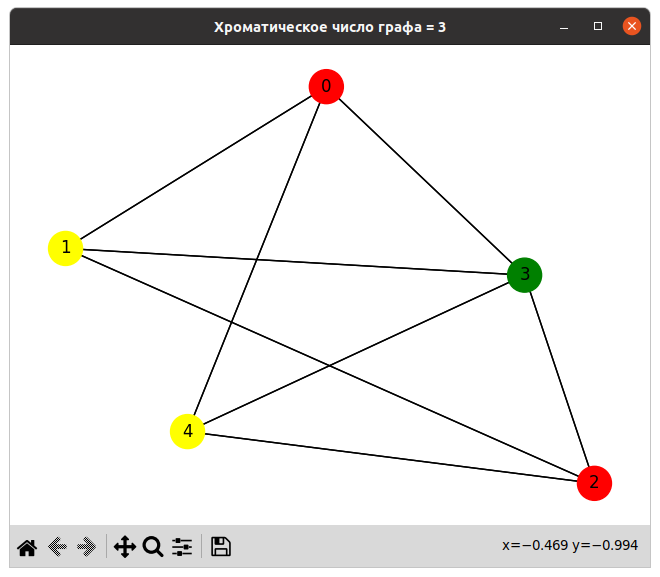


Рисунок 11 - Отрисовка окрашенного графа, считанного из файла, с указанием его хроматического числа

Однако, если мы введём некорректное число вершин (рис. 12) при генерации графа, то получим ошибку и прекращение работы алгоритма (рис. 13).

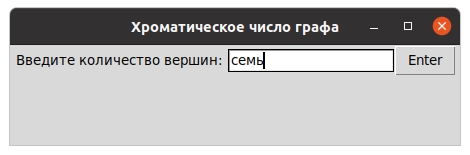


Рисунок 12 - Ввод некорректного значения

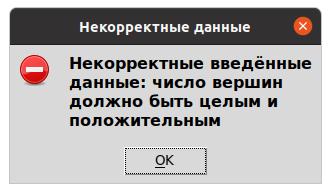


Рисунок 13 - Сообщение об ошибке

При вводе имени несуществующего файла с данными (рис. 14). Программа понимает, что его нет и предупреждает нас об этом (рис. 15).

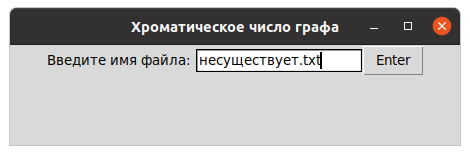


Рисунок 14 - Ввод имени несуществующего файла

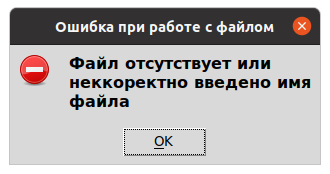


Рисунок 15 - Предупреждение об отсутствии файла

Программа также реагирует на некорректные данные, считанные из указанного файла (рис. 16, 17).

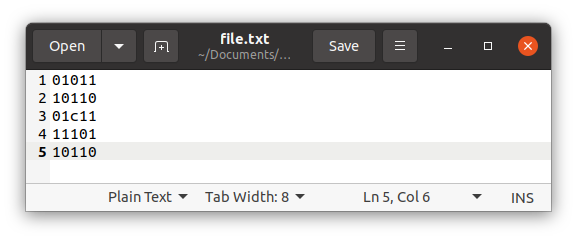


Рисунок 16 - Некорректно записанные данные в файл

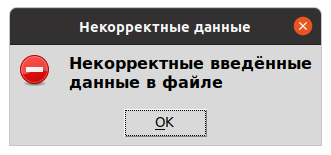


Рисунок 17 - Ошибка при чтении некорректных данных

**Тестирование**

Среды разработки Code::Blocks IDE и PyCharm Community Edition, предоставляет все средства, необходимые при разработке и отладке многомодульной программы. Тестирование проводилось как в процессе разработки, так и после завершения написания программы.

Ниже представлены результаты работы программы (рис. 18 - 21).

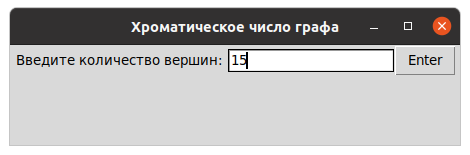


Рисунок 18 - Ввод числа вершин генерируемого графа

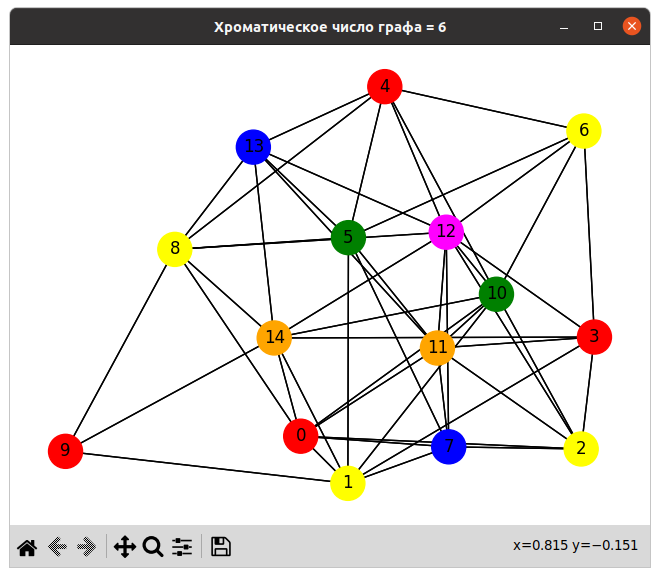


Рисунок 19 - Вывод раскрашенного графа и его хроматического числа

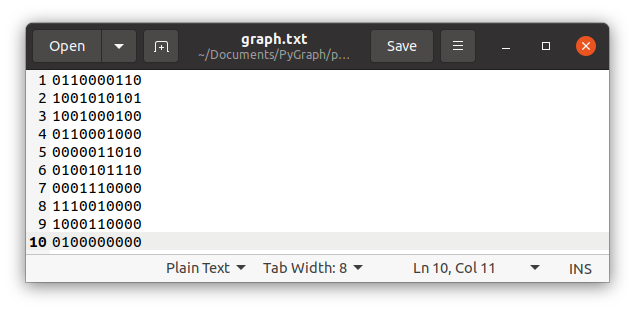


Рисунок 20 - Файл с матрицей смежности графа

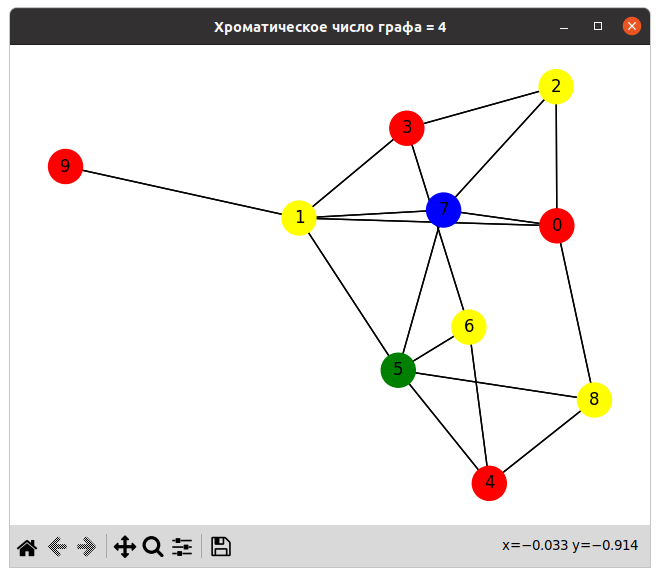


Рисунок 21 - Вывод раскрашенного графа и его хроматического числа

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Проверка запуска программы | Вывод информации о курсовом проекте | Верно |
| Проверка на  неправильный ввод числа вершин генерируемого графа | Программа должна оповестить пользователя об ошибке и прекратить работу алгоритма | Верно |
| Проверка на введение имени несуществующего файла | Программа должна оповестить пользователя об ошибке при вводе имени файла | Верно |
| Проверка на некорректные данные в файле | Программа должна оповестить пользователя об ошибке | Верно |
| Проверка на пустой файл с данными | Программа должна оповестить пользователя об ошибке | Верно |

Ниже представлены результаты тестирования программы (рис. 22 - 30).

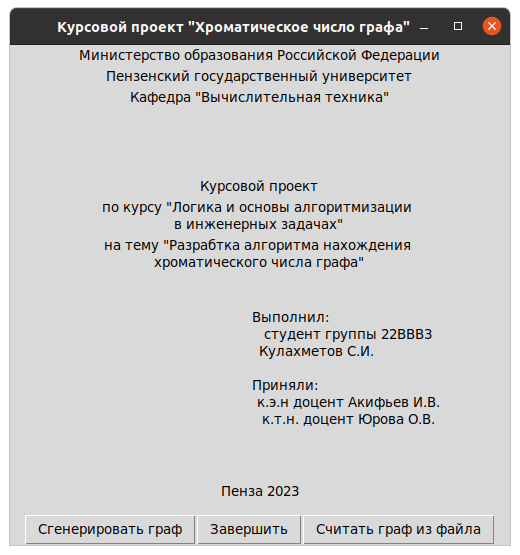


Рисунок 22 - Проверка запуска программы

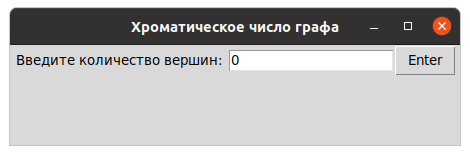


Рисунок 23 - Неправильный ввод числа вершин генерируемого графа

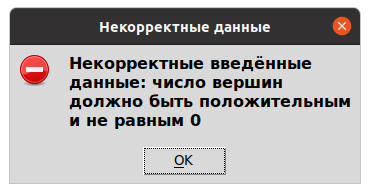


Рисунок 24 - Предупреждение об ошибке

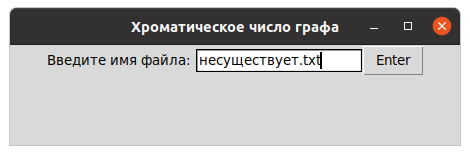


Рисунок 25 - Ввод имени несуществующего файла

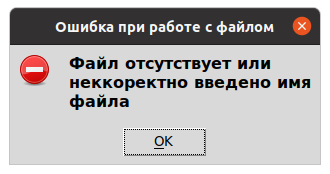


Рисунок 26 - Предупреждение об ошибке

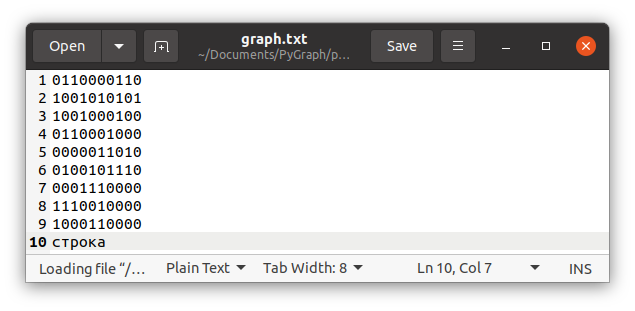


Рисунок 27 - Некорректные данные в файле

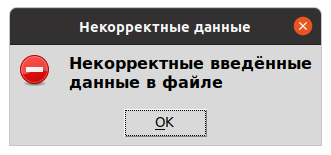


Рисунок 28 - Предупреждение об ошибке

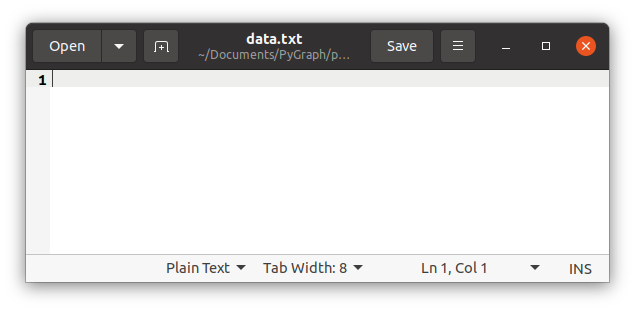


Рисунок 29 - Пустой файл, в котором должны быть данные

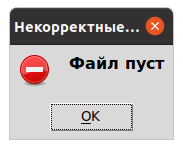


Рисунок 30 - Предупреждение об ошибке

**Ручной расчёт**

Для проверки корректности работы программы можно нарисовать граф самим, попробовать раскрасить и посчитать количество цветов (рис. 31). Затем сгенерировать граф при помощи программы (рис. 32).

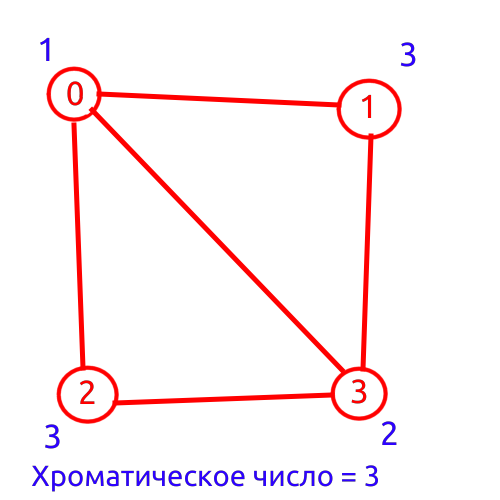


Рисунок 31 - Ручная отрисовка графа

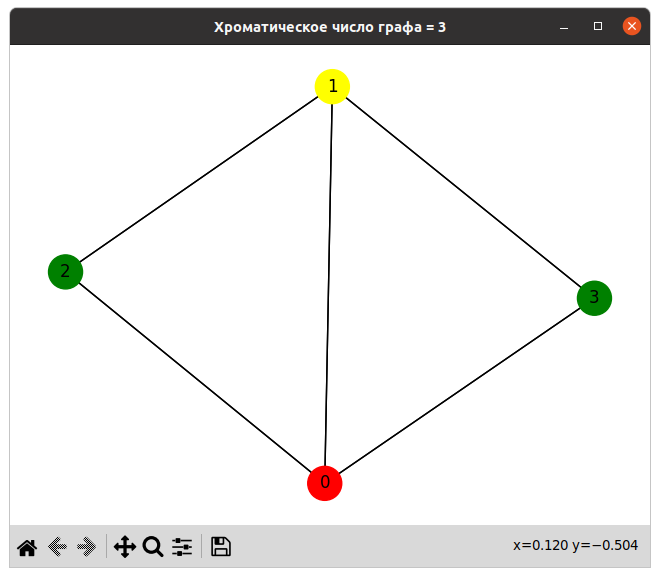


Рисунок 31 - Проверка программы

**Заключение**

В результате выполнения курсового проекта был реализован алгоритм поиска хроматического числа графа при помощи языков программирования Си и Python 3 в средах разработки Code::Blocks IDE и PyCharm Community Edition.

В ходе выполнения курсовой работы были получены навыки в разработке графических интерфейсов на языке Python 3 на базе фреймворка Tkinter, а также навыки написания динамических библиотек на языке Си в среде GNU/Linux.

Программа имеет графический интерфейс, что указывает на её соответствие современным требованиям.

**Список литературы**

1. Кольцов Д.М. «СИ НА ПРИМЕРАХ. Практика, практика и только практика». Изд. «Наука и Техника» 2019 г.
2. Оре О. «Графы и их применение»: Пер. с англ. 1965. 176 с.
3. Пол Дейтл, Харви Дейтл «С для программистов с введением в С11». Изд. «ДМК Пресс» 2014 г.
4. Спрингер Вильям “Гид по Computer Science”. Изд “Питер” 2020. 192 с.

**Приложение А.**

**Листинг программы.**

**Файл main.py**

from Graphs import \* # Модуль, реализующий основные функции программы

Window = Tk() # Инициализация виджета окна

Window.title('Курсовой проект "Хроматическое число графа"') # Заголовок окна

Window.geometry('500x500') # Размер окна в пикселях

MFrame = Frame() # Фрейм, отвечающий за текст

DFrame = Frame() # Фрейм, отвечающий за дату

Frame = Frame() # Фрейм, содержащий кнопки

# Функция завершает цикл обработки окна и ищет хроматическо число рандомно сгенерированного графа

def Random():

Window.destroy()

RandomGraph()

# Функция завершает цикл обработки окна и ищет хроматическо число графа, считанного из файла

def File():

Window.destroy()

FromFile()

# Метки текста и даты

lb0 = Label(MFrame, text='Министерство образования Российской Федерации')

lb1 = Label(MFrame, text='Пензенский государственный университет')

lb2 = Label(MFrame, text='Кафедра "Вычислительная техника"\n\n\n\n')

lb3 = Label(MFrame, text='Курсовой проект')

lb4 = Label(MFrame, text='по курсу "Логика и основы алгоритмизации \nв инженерных задачах"')

lb5 = Label(MFrame, text='на тему "Разрабтка алгоритма нахождения \nхроматического числа графа"\n\n')

lb6 = Label(MFrame, text='Выполнил:\t\n\tстудент группы 22ВВВ3\nКулахметов С.И.\n\nПриняли:\t\n\t'\

'к.э.н доцент Акифьев И.В.\n\tк.т.н. доцент Юрова О.В.')

lb7 = Label(DFrame, text='\n\n\nПенза 2023')

# Кнопки

BTN1 = Button(Frame, text="Сгенерировать граф", command=Random)

BTN2 = Button(Frame, text="Считать граф из файла", command=File)

btnExit = Button(Frame, text="Завершить", command=exit)

# Относительные позиции меток и кнопок

lb0.pack(side=TOP)

lb1.pack(side=TOP)

lb2.pack(side=TOP)

lb3.pack(side=TOP)

lb4.pack(side=TOP)

lb5.pack(side=TOP)

lb6.pack(side=RIGHT)

lb7.pack()

BTN1.pack(side=LEFT)

BTN2.pack(side=RIGHT)

# Позиции фреймов

btnExit.pack(side=RIGHT)

MFrame.pack(side=TOP)

DFrame.pack()

Frame.pack(side=BOTTOM)

# Цикл обработки событий окна

Window.mainloop()

**Файл Graphs.py**

import os # Модуль для взаимодействия с ОС и файловой системой

import networkx as nx # Модуль для рисования графа

import numpy as np # Модуль для математических операций

import matplotlib.pyplot as plt

from ctypes import \* # Модуль для поддержки типов данных языка С

from tkinter import \* # #одуль для отображения графического интерфейса

from tkinter.messagebox import showerror # Модуль для диалоговых окон

from Colors import \* # Модуль для преобразования числового ветора цветов в их строковые обозначения

# Подключение динамических библиотек, написанных на языке С, и реализующих основной алгоритм программы

Random\_Graph\_Matrix = CDLL('./RandGraph.so')

Read\_From\_File\_Matrix = CDLL('./ReadFromFile.so')

# Функция нахождения хроматического числа рандомно сгенерированного графа

def RandomGraph():

window = Tk() # Инициализация виджета окна

window.title("Хроматическое число графа") # Заголовок окна

window.geometry("450x100") # Размер окна в пикселях

frame = Frame() # Фрейм для элементов окна

entry = Entry(frame) # Поле ввода значения

# Функция получения количества вершин графа из поля ввода

def dialog():

global size # Глобальная переменная, отвечающая за количество вершин графа

#Обработка исключений

try: #В данном случае проверка на возможность конвертации в int

number=int(entry.get())

except ValueError: #Если значение - char, string или float, то выходит предупреждение

# Диалоговое окно, сообщающее об ошибке

showerror('Некорректные данные', 'Некорректные введённые данные: число вершин '\

'должно быть целым и положительным')

exit() # Выход из программы

size = number # Если всё хорошо, то сохраняем полученное число вершин

if size<1: # Если число вершин отрицательное, то граф генерировать не будем

showerror('Некорректные данные', 'Некорректные введённые данные: число '\

'вершин должно быть положительным и не равным 0')

exit()

elif size>20: # Слишком большое число вершин тоже не нужно, для корректности отображения

showerror('Некорректные данные', 'Слишком большое число вершин: '\

'предпочтительно генерировать не более 20 вершин')

exit()

window.destroy() # Выход из цикла обработки событий окна

btn = Button(frame, text="Enter", command=dialog) # Кнопка, ответственная за подтверждение ввода данных

lb = Label(frame, text="Введите количество вершин: ") # Метка с текстом возле поля ввода

# Относительные расположения элементов

lb.pack(side=LEFT)

entry.pack(side=LEFT)

btn.pack(side=RIGHT)

# Отображение фрейма

frame.pack()

# Цикл обработки событий окна

window.mainloop()

# Вызов функции из динамической библиоткети для генерации и раскрашивания графа

Random\_Graph\_Matrix.RandGraph(c\_int(size))

# Чтение данных из обменного файла

f = open('data.txt', 'r', encoding='utf-8')

data = f.read() # Чтение строки с данными

f.close() # Закрытие файла

k = int(0) # Инициализация удополнительное индексной перемнной

mas = [0] \* size # Матрица смежности, считываемая из файла

for i in range(size):

mas[i] = [0] \* size

colors = [0] \* size # Вектор цветов, считываемый из файла

# Считывание матрицы смежности из файла

for i in range(size):

for j in range(size):

mas[i][j] = int(data[k]) # Преобразование строки в int

k += 1

if data[k] == '\n':

k += 1

# Считывание вектора цветов из файла

for i in range(size):

colors[i] = int(data[k]) # Вектор цветов дополнительно дозаписан в файле для обмена данными

k += 1

# Инициализация вектора цветов для покраски графа

Spectral = ['white'] \* size

ChrNumber = int(0) #Хроматическое число

ChrNumber = ColorsInit(colors, Spectral, size) #Задание цветов для вершин

# непосредственное рисование графа

G = nx.DiGraph(np.matrix(mas)) # Рисование графа по матрице смежности при помощи модуля networkx

nx.draw(G, node\_color=Spectral, with\_labels=True, node\_size=600, arrows=False) # Опции рисования графа

Str = 'Хроматическое число графа = ' + str(ChrNumber) # Строка для отображения хроматического числа

fig = plt.gcf() # Получение рисуемого объекта

fig.canvas.manager.set\_window\_title(Str) # Назначение заголовка окна рисуемому объекту

plt.show() # Отображение окна с полем, содержащим граф

##############################################################################

def FromFile():

window = Tk()

window.title("Хроматическое число графа")

window.geometry("450x100")

frame = Frame(window)

entry = Entry(frame)

# Вложенная функция для обработки полученного имени файла

def dialog():

global s # Глобальная переменная, хранящая название файла с расширением

s = entry.get() # Получение имени файла из поля ввода

# Обработка исключений и ошибок

try: # Проверка на существование файла

fl = open(entry.get(), "r", encoding='utf-8')

except FileNotFoundError:

showerror('Ошибка при работе с файлом', 'Файл отсутствует или некорректно введено ' \

'имя файла')

exit()

# Проверка содержания файла

result = os.stat(entry.get())

if result.st\_size==0: # Если файл пуст, завершение работы алгоритма

showerror('Некорректные данные', 'Файл пуст')

exit()

fl.close()

file = open('filename.txt', 'w', encoding='utf-8') # Открытие обменного файла

file.write(entry.get()) # Запись имени целевого файла в обменный

file.close()

window.destroy()

btn = Button(frame, text="Enter", command=dialog)

lb = Label(frame, text="Введите имя файла: ")

lb.pack(side=LEFT)

entry.pack(side=LEFT)

btn.pack(side=RIGHT)

frame.pack()

window.mainloop()

'''

Вызов функции из динамической библиотеки,которая ответственна за раскрашивание графа,

считанного из целевого файла, получение возвращаемого значения количества вершин

'''

size=Read\_From\_File\_Matrix.ReadFromFile()

# Открыетие целевого файла

f = open(s, 'r', encoding='utf-8')

Data = f.read() # Чтение строки с данными

f.close()

Mas = [0] \* size # Матрица смежности, считываемая из файла

for i in range(size):

Mas[i] = [0] \* size

Colors = [0] \* size # Считываемый вектор цветов

spectral = ['white'] \* size # Вектор цветов для раскрашивания

k = int(0)

# Считывание матрицы смежности из файла

for i in range(0, size):

for j in range(0, size):

try: # Проверка на целочисленное значение ребра считываеого графа

Mas[i][j] = int(Data[k])

except ValueError: # Если значение - char, string или float, то выходит предупреждение

showerror('Некорректные данные', 'Некорректные введённые данные в файле')

exit()

# Если граф имеет неправильную форму, то индекс выйдет за пределы гипотетической матрицы

except IndexError:

showerror('Некорректные данные', 'Некорректные введённые данные в файле')

exit()

k += 1

try:

if Data[k] == '\n':

k += 1

except IndexError:

showerror('Некорректные данные', 'Некорректные введённые данные в файле')

exit()

# Считывание вектора цветов из файла

for i in range(0, size):

try: # Проверка на целочисленность значений вектора цветов

Colors[i] = int(Data[k])

except ValueError: # Если значение - char, string или float, то выходит предупреждение

showerror('Некорректные данные', 'Некорректные введённые данные в файле')

exit()

# Если вектор цветов сбит из-за неправильной формы графа, то индекс выйдет за пределы вектора

except IndexError:

showerror('Некорректные данные', 'Некорректные введённые данные в файле')

exit()

k += 1

# Инициализация хроматического числа графа и вызов функции преобразования вектора цветов

chrNumber = int(0)

chrNumber = ColorsInit(Colors, spectral, size)

# Непосредственное рисование графа

Gr = nx.DiGraph(np.matrix(Mas))

nx.draw(Gr, node\_color=spectral, with\_labels=True, node\_size=600, arrows=False)

Str = 'Хроматическое число графа = ' + str(chrNumber)

fig = plt.gcf()

fig.canvas.manager.set\_window\_title(Str)

plt.show()

**Файл Colors.py**

# Функция для преобразования числового вектора цветов в их строковые значения

def ColorsInit(colors, spectral, size):

for i in range(0, size):

if colors[i] == 0:

spectral[i] = 'white'

elif colors[i] == 1:

spectral[i] = 'red'

elif colors[i] == 2:

spectral[i] = 'yellow'

elif colors[i] == 3:

spectral[i] = 'green'

elif colors[i] == 4:

spectral[i] = 'blue'

elif colors[i] == 5:

spectral[i] = 'orange'

elif colors[i] == 6:

spectral[i] = 'magenta'

elif colors[i] == 7:

spectral[i] = 'cyan'

elif colors[i] == 8:

spectral[i] = 'pink'

elif colors[i] == 9:

spectral[i] = 'gray'

tmp = int(0) # Переменная для промежуточных значение

for i in range(0, size): # Поиск хроматического числа

if tmp<colors[i]:

tmp=colors[i]

return tmp # Возврат числа цветов

**Файл RandGraph.c**

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#define MAX\_SIZE 500

/\*

Функция, генерирующая псевдорандомом матрицу смежности графа,

хроматическое число которого требуется найти

\*/

void RandGraph(int size)

{

srand(time(NULL));

//int \*\*graph; // Указатель на матрицу смежности

//int \*colors; // Вектор цветов графа

int graph[MAX\_SIZE][MAX\_SIZE], colors[MAX\_SIZE], used\_colors[MAX\_SIZE];

/\*graph = (int \*\*)malloc(sizeof(int \*) \* size);

for(int i = 0; i < size; i++)

graph[i] = (int \*)malloc(sizeof(int) \* size);\*/

// Определение псевдорандомом смежных вершин

for(int i = 0; i < size; i++)

for(int j = i; j < size; j++)

{

graph[i][j] = rand() % 2;

graph[j][i] = graph[i][j];

if(i == j) graph[i][j] = 0;

}

//======================================================================

//colors = (int \*)malloc(sizeof(int) \* size);

//======================================================================

for (int i = 0; i < size; i++)

{

// Инициализация множества использованных цветов соседей

//int \*used\_colors = (int \*)malloc(sizeof(int) \* size);

for(int i = 0; i < size; i++)

used\_colors[i] = 0;

// Проходим по всем соседям текущей вершины и добавляем их цвета в множество

for (int j = 0; j < size; j++)

{

if (graph[i][j] && colors[j] != 0)

{

used\_colors[colors[j]] = 1;

}

}

// Выбор цвета для текущей вершины

for (int j = 1; j <= size; j++)

{

if (used\_colors[j] == 0)

{

colors[i] = j;

break;

}

}

//free(used\_colors);

}

//======================================================================

FILE \*file;

file = fopen("data.txt", "w");

for(int i = 0; i < size; i++)

{

for(int j = 0; j < size; j++)

{

fprintf(file, "%c", graph[i][j] + '0');

}

fprintf(file, "\n");

}

for(int i = 0; i < size; i++)

fprintf(file, "%c", colors[i] + '0');

fclose(file);

//free(colors);

//for(int i = 0; i < size; i++)

// free(graph[i]);

//free(graph);

}

**Файл ReadFromFile.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_SIZE 500

int ReadFromFile()

{

char filename[80];

int graph[MAX\_SIZE][MAX\_SIZE];

int size;

int colors[MAX\_SIZE], used\_colors[MAX\_SIZE];

//=======================================================

FILE \*file, \*name;

int count = 0, k = 0;

char c, arr[MAX\_SIZE \* 2];

name = fopen("filename.txt", "r");

fscanf(name, "%s", filename);

fclose(name);

file = fopen(filename, "r");

while((c = fgetc(file)) != EOF)

count++;

rewind(file);

for(int i = 0; i < count; i++)

arr[i] = fgetc(file);

size = 0;

while(arr[size] != '\n')

size++;

for(int i = 0; i < size; i++)

{

for(int j = 0; j < size; j++)

{

graph[i][j] = arr[k];

k++;

if(arr[k] == '\n') k++;

graph[i][j] = graph[i][j] - 48;

}

}

fclose(file);

//=======================================================

for(int i = 0; i < size; i++)

colors[i] = 0;

//=======================================================

for (int i = 0; i < size; i++)

{

// инициализация множества использованных цветов соседей

for(int i = 0; i < size; i++)

used\_colors[i] = 0;

// проходим по всем соседям текущей вершины и добавляем их цвета в множество

for (int j = 0; j < size; j++)

{

if (graph[i][j] && colors[j] != 0)

{

used\_colors[colors[j]] = 1;

}

}

// выбор цвета для текущей вершины

for (int j = 1; j <= size; j++)

{

if (used\_colors[j] == 0)

{

colors[i] = j;

break;

}

}

}

//========================================================

file = fopen(filename, "a");

for(int i = 0; i < size; i++)

fprintf(file, "%c", colors[i] + 48);

fclose(file);

return size;

}