**Лабораторная работа №4**

*Задача о раскраске графа:* для заданного числа красок раскрасить вершины обыкновенного графа так, чтобы смежные вершины были раскрашены в разные цвета или определить, что это невозможно.

Для решения задачи о раскраске графа использую метод поиска с возвратом.

**Код**

Конструктор класса Graph

class Graph:  
 def \_\_init\_\_(self, vertices):  
 self.V = vertices # Количество вершин  
 self.graph = [[] for \_ in range(vertices)] # Список смежности

- *\_\_init\_\_* - это конструктор класса, который инициализирует объект графа.

- *self.V* хранит количество вершин в графе.

- *self.graph* - это список смежности, который представляет граф как массив списков. Каждый элемент массива соответствует вершине и содержит список её соседей.

Метод добавления рёбер

def add\_edge(self, u, v):  
 self.graph[u].append(v)  
 self.graph[v].append(u)

- *add\_edge* добавляет рёбра между вершинами u и v. Поскольку граф неориентированный, добавляю v в список соседей u и наоборот.

Проверка безопасности раскраски

def is\_safe(self, v, color, c):  
 for neighbor in self.graph[v]:  
 if color[neighbor] == c:  
 return False  
 return True

- метод *is\_safe* проверяет, безопасно ли окрасить вершину v цветом c.

- проходит по всем соседям вершины v и проверяет, нет ли у них того же цвета. Если есть, возвращает False, иначе - True.

Вспомогательный метод для раскраски графа

def graph\_coloring\_util(self, m, color, v):  
 if v == self.V:  
 return True

- *graph\_coloring\_util* - это рекурсивный метод, который пытается раскрасить граф.

- если v равно количеству вершин, это означает, что все вершины были успешно раскрашены, и метод возвращает True.

for c in range(1, m + 1):  
 if self.is\_safe(v, color, c):  
 color[v] = c

- в этом цикле мы перебираем доступные цвета от 1 до m.

- если текущий цвет безопасен для вершины v, он назначается этой вершине.

if self.graph\_coloring\_util(m, color, v + 1):  
 return True  
  
color[v] = 0

- после назначения цвета вызывается рекурсивный вызов для следующей вершины (v + 1).

- если раскраска не удалась (возвращает False), происходит возврат - цвет у вершины сбрасывается на 0.

return False

- если ни один цвет не подходит для данной вершины, метод возвращает False.

Метод раскраски графа

def graph\_coloring(self, m):  
 color = [0] \* self.V  
 if not self.graph\_coloring\_util(m, color, 0):  
 print("Невозможно раскрасить граф с", m, "цветами.")  
 return False

- *graph\_coloring* инициализирует массив цветов для всех вершин.

- если раскраска не удалась (метод возвращает False), выводится сообщение об ошибке.

print("Решение раскраски графа:", color)  
self.generate\_graphviz\_file(color)  
return True

- если раскраска успешна, выводится массив цветов и вызывается метод для генерации файла Graphviz.

Генерация файла Graphviz

def generate\_graphviz\_file(self, color):  
 with open("graph\_coloring.dot", "w") as f:  
 f.write("graph G {\n")

- метод *generate\_graphviz\_file* создает файл *graph\_coloring.dot*, который будет содержать описание графа в формате Graphviz.

- открывается файл для записи.

for v in range(self.V):  
 f.write(f' {v} [label="{v}", style=filled, fillcolor="{self.get\_color\_name(color[v])}"];\n')

- для каждой вершины записывается строка с её номером и цветом. Цвет определяется с помощью метода *get\_color\_name*.

for u in range(self.V):  
 for v in self.graph[u]:  
 if u < v: # Чтобы не дублировать рёбра  
 f.write(f' {u} -- {v};\n')

- далее записываются рёбра между вершинами. Условие if u < v предотвращает дублирование рёбер.

f.write("}\n")

- закрывающая фигурная скобка завершает описание графа.

Получение имени цвета

def get\_color\_name(self, color):  
 # Возвращает имя цвета в зависимости от номера цвета  
 colors = ["white", "red", "green", "blue", "yellow", "orange", "purple", "cyan", "magenta"]  
 return colors[color % len(colors)]

- метод *get\_color\_name* возвращает имя цвета в зависимости от номера цвета. Используется массив с названиями цветов.

Основная часть программы

if \_\_name\_\_ == "\_\_main\_\_":  
 # Создание графа  
 g = Graph(9) # Пример графа с 9 вершинами  
 edges = [  
 (0, 1), (0, 2), (0, 3),  
 (1, 2), (1, 4), (1, 5),  
 (2, 3), (2, 6),  
 (3, 7),  
 (4, 5), (4, 8),  
 (5, 6),  
 (6, 7),  
 (7, 8)  
 ]  
  
 for u, v in edges:  
 g.add\_edge(u, v)  
  
 m = 4 # Количество цветов для раскраски  
  
 print("Решение задачи о раскраске графа методом поиска с возвратом:")  
 g.graph\_coloring(m)

- в этом блоке создается объект графа с 9 вершинами.

- определяются рёбра между вершинами.

- после добавления рёбер вызывается метод для раскраски графа с заданным количеством цветов (m = 4).

**Результат**

