МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з Основ програмування
(назва дисципліни)

на тему: «Розфарбування графів»

Студента 1 курсу, групи ІП-14

Нікуліна Павла Юрійовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник Головченко Максим Миколайович
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____

Національна оцінка _____

Члени комісії

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-14 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Нікуліна Павла Юрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи «Розфарбування графів»
2. Строк здачі студентом закінченої роботи 12.06.2022
3. Вихідні дані до роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Дата видачі завдання 18 лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	03.05.2022	
4.	Розробка сценарію роботи програми	04.05.2022	
6.	Узгодження сценарію роботи програми з керівником	04.05.2022	
5.	Розробка (вибір) алгоритму рішення задачі	04.05.2022	
6.	Узгодження алгоритму з керівником	04.05.2022	
7.	Узгодження з керівником інтерфейсу користувача	05.05.2022	
8.	Розробка програмного забезпечення	06.05.2022	
9.	Налагодження розрахункової частини програми	06.05.2022	
10.	Розробка та налагодження інтерфейсної частини програми	07.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	25.05.2022	
12.	Тестування програми	26.05.2022	
13.	Підготовка пояснювальної записки	05.06.2022	
14.	Здача курсової роботи на перевірку	12.06.2022	
15.	Захист курсової роботи	15.06.2022	

Студент _____
(підпис)

Керівник _____ Головченко Максим Миколайович
(підпис) (прізвище, ім'я, по батькові)

"_____" _____ 2022 р.

АНОТАЦІЯ

Пояснювальна записка до курсової: 58 сторінок, 16 рисунків, 11 таблиць.

Об’єкт дослідження: розфарбування графів.

Мета роботи: дослідження методів розфарбування неорієнтованих графів (жадібний та бектрегінг з двома евристиками), розробка програмного забезпечення для побудування графа та його розфарбування, в залежності від обраного метода.

Опановано розробку програмного забезпечення з використанням ООП, приведені змістовні постановки задач, відповідні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація побудови і розфарбування неорієнтованого графа різними методами.

НЕОРІЄНТОВАНІ ГРАФИ, РОЗФАРБУВАННЯ ГРАФІВ, ЖАДІБНИЙ МЕТОД, ПОШУК З ПОВЕРНЕННЯМ, СТЕПЕНЕВА ЕВРИСТИКА, ЕВРИСТИКА MRV.

ЗМІСТ

ВСТУП

Дана курсова робота присвячена створенню програми для побудови та розфарбування неорієнтованого графа. Буде опановано три методи розфарбування: жадібний та пошук з поверненням (бектрекінг) з MRV та степеневою евристиками. Для створення такої програми буде використано парадигму ООП з принципами поліморфізму та інкапсуляціїї у мові програмування Python. Буде розроблено графічний інтерфейс з інтерактивними елементами для користувача.

Основна частина курсової роботи буде розділена на 7 розділів.

У першому розділі буде описана головна задача, поставлені підзадачі та визначені способи для їх реалізації

Другий розділ буде присвячено теоретичним відомостям про те, що таке неорієнтований граф, що означає його розфарбування, та як працюють необхідні для цього методи.

У третьому розділі буде описано загальний алгоритм програми, а також алгоритми підзадач.

У четвертому розділі буде описано методи частин програмного забезпечення (стандартні та базові), а також наведено діаграму класів.

П’ятий розділ допоможе користувачеві з’ясувати як потрібно працювати з програмою, які дані потрібно буде ввести для побудування графа та які дані буде отримано. Також буде наведено таблицю з системними вимогами, необхідними для запуску програми.

Останній розділ буде присвячено аналізу тестування програмного забезпечення. У цьому розділі буде наведено докази правильності алгоритмів розфарбування графа, буде виявлено їх складність та обрано найоптимальніший метод.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде створювати неорієнтований граф на базі списку зв’язків вершин, цього графа.

Першим вхідним даним є число, що описує кількість вершин у графі, у наступному форматі:

amount

За допомогою графічного інтерфейсу можна додати список суміжностей для поточної вершини або перейти до наступної. Для кожної вершини окремо вводиться список вершин, з якими вона суміжна, у форматі:

A B C ...

, де A, B та C – це вершини, які суміжні з вершиною *X*. Якщо, наприклад вершина C була оголошена суміжною до вершини A, то введення вершини A у список суміжностей вершини C є необов’язковим. Якщо залишити це поле пустим, то жодної вершини не буде додано до списку. Створений граф буде записано у бінарний файл, який буде видалено по завершенню програми.

На наступному етапі обрається один з трьох шляхів: графічне відображення введеного графа, перейти до методів його розфарбування або закрити програму.

На сторінці вибору методу розфарбування є три варіанти: жадібний, бектрегінг з використанням степеневої евристики та бектрекінг з використанням евристики MRV. На цій самій сторінці присутня навігація (назад/закрити).

При виборі одного з методу розфарбування використовуються дані графа з файлу та проводяться розрахунки. Виводиться послідовність вершин, яка була використана для цього методу у форматі

Sequence of vertices to color (<method_name>)

B A D C

, і сам розфарбований граф у графічному вигляді.

2 ТЕОРИТИЧНІ ВІДОМОСТІ

Граф – це сукупність об’єктів (вершин) зі зв’язками (ребрами) між ними.

Ребра графу можуть бути *напрямленими* або *ненапрямленими*. Граф першого типу називається *неорієнтованим* графом, , тоді як граф другого типу називається *орієнтованим* графом. У цій роботі буде розглянуто неорієнтований граф.

Граф можна задавати декількома способами: графічно, списком ребер, списком суміжностей або матрицею суміжностей. На графічному відображенні графа змальовуються його вершини, поєднані лініями (ребра графа).

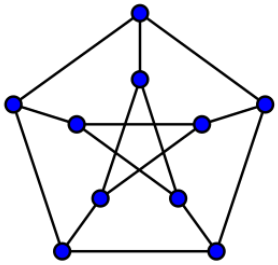


Рисунок 2.1 (приклад графічного зображення графа)

У позначаються усі зв’язки (ребра) графу.

1	I, II
2	I, III
3	II, IV
4	I, V
5	II, VI
6	III, IV
7	III, V
8	IV, VI
9	V, VII
10	VI, VII

Рисунок 2.2 (приклад списку ребер графа)

У *списку суміжностей* для кожної окремої вершини позначаються усі вершини, для яких вона є суміжною.

1	2, 4
2	3, 4
3	2
4	1, 3

Рисунок 2.3 (приклад списку суміжностей графа)

У матриці суміжностей позначаються усі з’єднання (ребра) для кожної вершини. Рядок – це вершина *X*, стовпці – вершини, які є суміжні до *X*.

Елементами матриці є нулі та одиниці (0 – немає суміжностей, 1 – суміжність є).

R=

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
x_1	1	1	0	1	1	0	0
x_2	0	1	0	1	1	0	0
x_3	0	0	1	1	1	0	0
x_4	0	0	0	1	1	0	0
x_5	0	0	0	0	1	0	0
x_6	0	0	1	1	1	1	1
x_7	0	0	1	1	1	1	1

Рисунок 2.4 (приклад матриці суміжностей графа)

Під час виконання даної курсової роботи буде використано графічне зображення графа, його список ребер, список суміжностей (буде переведено у список ребер) та матриця суміжностей.

Розфарбуванням графу називають таке приписування кольорів (або інших атрибутів) вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають *хроматичним числом*. Величина хроматичного числа залежить від типу графа, але ніколи не буде більше кількості вершин цього ж графа. У даній роботі буде використано два алгоритми розфарбування графів: жадібний метод та пошук з поверненням (бектрекінг).

Жадібний метод фарбує вершини у послідовності, зазначеній до початку фарбування. Такою послідовністю може бути, наприклад за абеткою або за порядком додавання вершин. У курсовій роботі буде використано послідовність за абеткою і порядком додавання одночасно (вершини додаватимуться за абеткою).

Пошук с поверненням або *бектрекінг* полягає у тому, щоб вибрати наступну вершину вже в процесі фарбування. Критерій вибору визначається евристикою, яких у цій роботі буде використано дві: MRV та степенева евристики). *Степенева евристика* визначає критерієм ступінь суміжностей вершини (кількість суміжних вершин дорівнюють ступеню). *Евристика MRV (minimal remaining value)* обирає наступну вершину за кількістю можливих кольорів для її фарбування (обирається найменше число)

3 ОПИС АЛГОРИТМІВ

Перелік усіх основних змінних та їхні призначення наведено у таблиці 3.1

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
vertices	Список вершин графа
edges	Список ребер графа
edges_list	Список ребер вершини
matrix	Матриця суміжностей графа
colors	Список доступних кольорів
greedy_coloring/bd_coloring/ mrv_coloring	Послідовність додавання кольору для кожного алгоритма
max_degree	Максимальний ступінь вершини
vertex_degree	Сума рядку matrix відповідної вершини
index/indices	Індекс/Список індексів вершин
bd_sequence/mrv_sequence	Порядок розфарбування вершин
min_colors	Мінімальна кількість доступних кольорів
mrv_available	Словник кількості доступних кольорів для вершин

3.1 Загальний алгоритм

1. ПОЧАТОК
2. Отримати кількість вершин n.
3. Створити список vertices з n вершин (великі латинські літери за англійською абеткою)
4. ЦИКЛ проходу за елементом vertex у списку vertices
    3.1 ЯКЩО вибрано додавання ребер до вершини vertex, (3.2), ІНАКШЕ, перейти до наступного проходу цикла
5. ЯКЩО обрано зобразити граф, вивести графічне відображення графа, використовуючи список ребер edges, ІНАКШЕ ЯКЩО обрано розфарбувати граф,

4.1 ЯКЩО обрано жадібний метод, (3.7), ІНАКШЕ ЯКЩО обрано метод бектрегінгу (евристика MRV), (3.9), ІНАКШЕ ЯКЩО обрано метод бектрегінгу (степенева евристика), (3.8), ІНАКШЕ ЯКЩО вибрано повернутись, (3.1.5), ІНАКШЕ ЯКЩО вибрано закрити програму, КІНЕЦЬ

6. ЯКЩО обрано завершити програму, КІНЕЦЬ
7. КІНЕЦЬ

3.2 Алгоритм додавання списку суміжностей до вершини

1. ПОЧАТОК
2. Отримати поточну вершину vertex з попередньої функції
3. Отримати рядок ребер у форматі <вершина$_1$> <вершина$_2$> … <вершина$_n$> для vertex
4. Розбити отриманий рядок за пробілами на список edges_list
5. ЦИКЛ проходу за елементом vertex_edge списку edges_list

4.1 ЯКЩО [vertex, vertex_edge] немає у списку суміжностей або [vertex_edge, vertex] немає у списку суміжностей, додати [vertex, vertex_edge] до списку суміжностей edges

4.2 Оновити матрицю суміжностей matrix. matrix[номер поточної вершини у vertices][vertex_edge]

6. КІНЕЦЬ

3.3 Алгоритм додавання кольору за послідовністю

1. ПОЧАТОК
2. ЦИКЛ проходу за елементом (curr_vertex) у списку вершин

1.1 Доступні кольори set_color = кольорам вершини colors[curr_vertex]

1.2 Послідовність кольорів colors_seq[індекс елемента vertex у списку вершин] = set_color

1.3 Список поточних ребер для вершини curr_edges = списку з

рядку matrix відповідної вершини

1.4 ЦИКЛ проходу за елементом i у кількості елементів списку curr_edges

1.4.1 ЯКЩО curr_edges[i] (суміжність vertex та поточної вершини) дорівнює 1 ТА set_color[0] (перший доступний колір) є у списку кольорів поточної вершини ТА vertex не дорівнює поточній вершині, прибрати set_color[0] з colors[curr_vertex]

3. ПОВЕРНУТИ colors_seq
4. КІНЕЦЬ

3.4 Алгоритм жадібного методу

1. ПОЧАТОК
2. Створити словник colors з n ключів, що є назвами вершин і присвоїти їм списки з n кольорів
3. greedy_coloring = (3.3)
4. Відобразити граф, кольори вершин якого були додані за послідовністю greedy_coloring
5. КІНЕЦЬ

3.5 Алгоритм методу бектрегінг (степенева евристика)

1. ПОЧАТОК
2. Створити словник colors з n ключів, що є назвами вершин і присвоїти їм списки з n кольорів
3. ЦИКЛ проходу по кількості вершин n

1.1 Максимальний ступінь вершини max_degree = 0

1.2 ЦИКЛ проходу за елементом i у кількості вершин n

1.2.1 ЯКЩО i немає у списку індексів indices ТА vertex_degree більше max_degree,

1.2.1.1 max_degree = vertex_degree

1.2.1.2 index = i

1.3 Додати index до indices

1.4 Додати до bd_sequence (послідовність вершин для фарбування) вершину зі списку вершин за номером index

2. bd_coloring = (3.3)
3. Відобразити граф, кольори вершин якого були додані за послідовністю bd_coloring
4. КІНЕЦЬ

3.6 Алгоритм методу бектрегінг (евристика MRV)

1. ПОЧАТОК
2. Створити словник colors з n ключів, що є назвами вершин і присвоїти їм списки з n кольорів
3. Створити словник mrv_available з n ключів, що є назвами вершин і присвоїти їм n (кількість можливих кольорів для вершини на початку фарбування)
4. ЦИКЛ проходу за елементом у кількості вершин n

3.1 Мінімальна кількість доступних кольорів min_colors = 20 (максимальна кількість вершин та кольорів)

3.2 ЦИКЛ проходу за елементом і у кількості вершин n

3.2.1 ЯКЩО і немає у списку індексів indices ТА mrv_available[vertices[i]] менше min_colors,

3.2.1.1 min_colors = mrv_available[vertices[i]]

3.2.1.2 index = i

3.3 Додати index до indices

3.4 Додати до mrv_sequence (послідовність вершин для фарбування) вершину зі списку вершин за номером index

3.5 ЦИКЛ проходу за елементом edge у edges

3.5.1 ЯКЩО поточна вершина є у edge ТА поточна вершина дорівнює edge[0] ТА edge[1] немає у mrv_sequence,

mrv_sequence[edge[1]] відняти 1, ІНАШКЕ ЯКЩО поточна вершина є у edge ТА поточна вершина дорівнює edge[1] ТА edge[0] немає у mrv_sequence, mrv_sequence[edge[0]] відняти 1

5. mrv_ coloring = (3.3)
6. Відобразити граф, кольори вершин якого були додані за послідовністю mrv_coloring
7. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

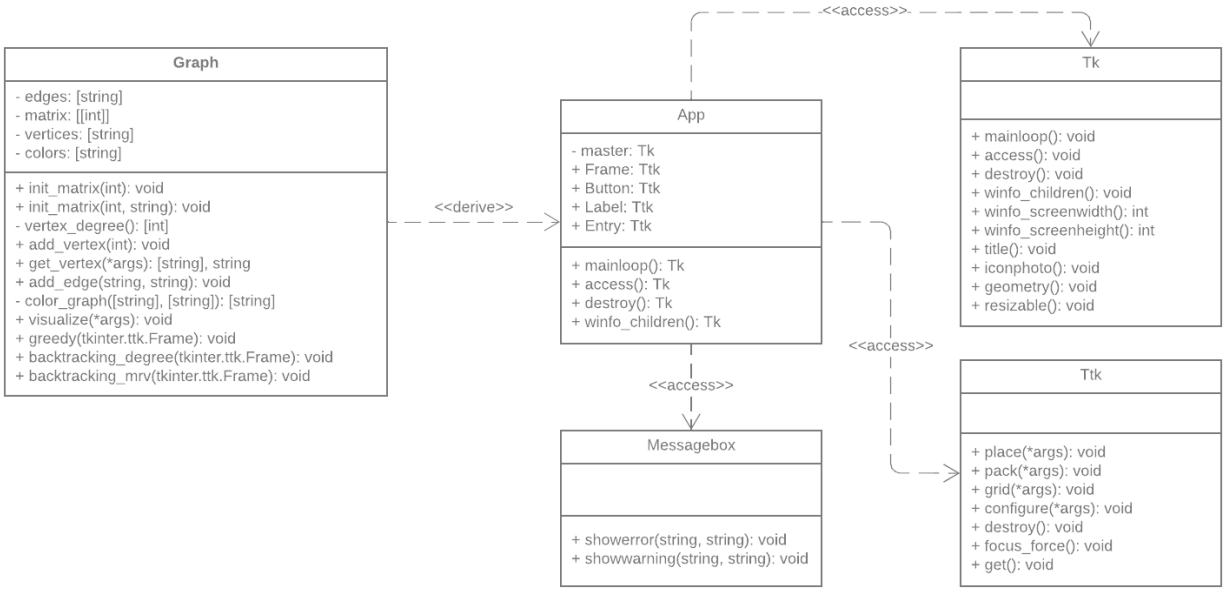


Рисунок 4.1 – Діаграма класів

4.2 Опис методів частин програмного забезпечення

4.2.1 Стандартні методи

У таблиці 4.1 наведено стандартні методи, використані при розробці програмного забезпечення.

Таблиця 4.1 – Стандартні методи

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	Python Built-in	chr	Перетворити цілий тип на символьний	Integer	Character
2	Python Built-in	int	Перетворити тип на цілий	String	Integer

Продовження таблиці 4.1

3	Python Built-in	range	Послідовність цілих чисел	Start = integer (default 0) Stop = integer Step = integer (default 1)	List of integers
4	Python Built-in	open	Відкриття файлу	File_name = string Mode = string (default ‘r’)	File object
5	Python Built-in	append	Додавання елемента до списку/словнику	Element = any type	-
6	Python Built-in	len	Довжина списка/словника	List/ dictionary	Довжина списку/словнику
7	Python Built-in	index	Індекс елемента у списку	Item in list = any type	Integer (index of item)
8	Python Built-in	remove	Прибрати елемент зі списку	Item in list= any type	-

4.2.2 Користувацькі методи

У таблиці 4.2 наведено користувацькі методи, створені при розробці програмного забезпечення.

Таблиця 4.2 – користувацькі методи

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	Graph	__init__	Конструктор класа Graph	-	-
2	Graph	init_matrix	Ініціалізація матрці суміжностей	Integer	-
3	Graph	set_matrix	Оновлення матрці суміжностей	Integer, string	-
4	Graph	__vetrtices_degree	Знаходження степеню вершини	-	List of integers
5	Graph	add_vertex	Додати вершини до списку вершин	String	-
6	Graph	get_vertex	Повернути вершину/список вершин	*args	String/list of strings
7	Graph	add_edge	Додати ребро до списку ребер	String, string	-
8	Graph	check_isolated	Перевірка вершин на ізольованість	-	List of strings
9	Graph	__color_graph	Розфарбувати граф	List of string, list of integers	List of strings
10	Graph	visualize	Відобразити граф графічно	*args	-
11	Graph	greedy	Жадібний метод розфарбування	Tkinter Ttk	-

Продовження таблиці 4.2

12	Graph	backtracking_degree	Метод бектрегінку (степенева евристика) для розфарбування	Tkinter Ttk	-
13	Graph	backtracking_mrv	Метод бектрегінку (евристика MRV) для розфарбування	Tkinter Ttk	-
14	NetworkX Graph	add_node	Додати вершину у граф	String	-
15	NetworkX Graph	add_edges_from	Додати список ребер для графічного виведення графу	List of (list of strings)	-
16	NetworkX	draw_networkx	Створити графічне зображення графа	nx.Graph, node_color (optionally) = list if strings	-
17	Matplotlib	gcf().canvas.set_window_title	Задати назву вікну з графічним зображенням графа	String	-
18	Matplotlib	show	Показати граф	-	-
19	Pickle	dump	Завантажити дані у файл	Any type, file object	-
20	Pickle	load	Дістати дані з файлу	File object	Any type
21	Tk	winfo_screenwidth	Отримати довжину екрану користувача	-	Integer

Продовження таблиці 4.2

22	Tk	winfo_screen height	Отримати висоту екрану користувача	-	Integer
23	Tk	winfo_childr en	Отримати список елементів в інтерфейсі	-	List of Ttk
24	Tk	title	Встановити назву вікна графічного інтерфейсу	String	-
25	Tk	iconphoto	Встановити іконку вікна графічного інтерфейсу	Bool, PhotoImage(fil e = string)	-
26	Tk	geometry	Встановити розмір вікна графічного інтерфейсу	String	-
27	Tk	resizable	Визначити можливість змінювати розмір вікна	Width = bool, height = bool	-
28	Tk	mainloop	Запустити роботу графічного інтерфейсу	-	-
29	Tk	destroy	Знищити об’єкт графічного інтерфейсу	-	-
30	App	winfo_childr en	Отримати список елементів в інтерфейсі	-	Tk

Продовження таблиці 4.2

31	App	mainloop	Запустити роботу графічного інтерфейсу	-	Tk
32	App	destroy	Знищити об’єкт графічного інтерфейсу	-	Tk
33	Ttk	pack	Відобразити віджет у інтерфейсі	pady = integer, padx = integer, ipady = integer, (optionally, but not used)	-
34	Ttk	place	Відобразити віджет у інтерфейсі	Anchor = string, relx = float, rely = float, (optionally, but not used)	-
35	Ttk	grid	Відобразити віджет у інтерфейсі	pady = integer, padx = integer, ipady = integer, row = integer, column = integer, (optionally, but not used)	-
36	Ttk	focus_force	Зосередити увагу системи на віджет	-	-
37	Ttk	destroy	Знищити віджет	-	-
38	Ttk	configure	Змінити налаштування віджету	text = string, (optionally, but not used)	-

Продовження таблиці 4.2

39	messagebox	showerror	Відобразити вікно повідомлення у стилі «Помилка»	String, string	-
40	messagebox	showwarning	Відобразити вікно повідомлення у стилі «Попередження»	String, string	-
41	Os	path.exists	Перевірити існування файлу	String	-
42	Os	remove	Видалити файл	String	-

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

Складемо план тестування програмного забезпечення, який включає у собі тестування основного функціоналу програми та перевірку реакцій на виключні ситуації:

а) Тестування перевірки правильності введеного значення кількості вершин
   1) Тестування введення кількості вершин, яке не є числом
   2) Тестування введення кількості вершин, що не підпадає під умову задачі

б) Тестування перевірки правильності введених значень суміжних вершин
   1) Тестування введення вершин, відсутніх у списку
   2) Тестування введення набору символів замість назв вершин

в) Тестування відсутності з’єднань між вершинами графу
   1) Тестування відсутності з’єднань для окремої вершини
   2) Тестування відсутності з’єднань для всіх вершин графу

г) Тестування алгоритмів розфарбування графа
   1) Тестування коректності роботи алгоритму жадібного методу
   2) Тестування коректності роботи алгоритму методу бектрекінг зі степеневою евристикою
   3) Тестування коректності роботи алгоритму методу бектрекінг з евристикою MRV

5.2 Приклади тестування

Проведемо тестування програмного забезпечення згідно з розробленим планом. Запишемо мету, початковий стан програми, вхідні дані, схему проведення, очікуваний результат і стан програми після проведення випробувань в окрему таблицю для кожного тесту.

Результати тестування основного функціоналу програмного забезпечення,

а також деяких аспектів та виняткових випадків наведено у таблицях 5.1 – 5.6. Додаткова інформація для пояснення буде наведена на рисунках 5.1 – 5.12 після таблиць.

Таблиця 5.1 - Тестування перевірки правильності введеного значення кількості вершин

Мета тесту	Перевірити реакцію програми на введення символа, що не є числом, у поле кількості вершин
Початковий стан програми	Відкрите вікно програми з полем для введення кількості вершини
Вхідні дані	Тест 1: abc Тест 2: 0 Тест 3: 69
Схема проведення тесту	Ввести дані у поле для введення кількості вершин
Очікуваний результат	Повідомлення про помилку Тест 1: «Введіть число від 1 до 20» Тест 2: «Введіть принаймні 1 вершину» Тест 3: «Введіть не більше 20 вершин»
Стан програми після проведення випробувань	Відкрите вікно для введення нових значень

Таблиця 5.2 - Тестування перевірки правильності введених значень суміжних вершин (введені дані не є вершинами у списку або є набором символів)

Мета тесту	Перевірити реакцію програми на введення суміжних вепшин, яких немає у списку вершин
Початковий стан програми	Відкрите вікно програми з полем для введення суміжних вершин
Вхідні дані	Доступні вершини: A, B, C, D Введення суміжностей для вершини A Тест 1: b C d O Тест 3: B pOx D 3@ Тест 3: B C D
Схема проведення тесту	Введення даних у поле суміжних вершин
Очікуваний результат	Список суміжнсоті для вершини A Тест 1: C Тест 2: B, D Тест 3: B, C, D
Стан програми після проведення випробувань	Перехід до вікна введення ребер наступної вершини (для останньої: вибір між графічним зображенням та розфарбуванням)

Таблиця 5.3 - Тестування відсутності з’єднань між вершинами графу (для окремої ізольованої вершини та для пустого графа)

Мета тесту	Перевірити реакцію програми на введення даних, що не є вершинами зі списку або є набором символів
Початковий стан програми	Відкрите вікно вибору між графічним зображенням та розфарбуванням
Вхідні дані	Ізольовані вершини у графі Тест 1: Ізольована вершина C у графі з вершинами A, B, C, D Тест 2: Ізольовані всі вершини (A, B, C, D) у графі
Схема проведення тесту	Пропуск вершин(и) на етапі введення ребер, вибір графічного відображення графа
Очікуваний результат	Тест 1: виведення графа з інформацією про ізольовані вершини Тест 2: виведення повідомлення про те, що граф пустий
Стан програми після проведення випробувань	Неможливість виведення та розфарбування графа без його скидання

Таблиця 5.4 - Тестування алгоритмів розфарбування графа (жадібний метод)

Мета тесту	Перевірити алгоритм розфарбування жадібним методом
Початковий стан програми	Відкрите вікно з вибором методу розфарбування графа
Вхідні дані	Вибір жадібного методу у графічному інтерфейсі для графу з вершинами A, B, C, D Список ребер: AB, AC, BC, DA Доступні кольори (маскимум 4): червоний, зелений, синій, жовтий
Схема проведення тесту	Розфарбувати граф використовуючи алфавітну послідовність вершин. Призначити кольори вершинам так, щоб сусідні вершини мали різний колір
Очікуваний результат	Послідовність вершин для розфарбування: A, B, C, D Призначені кольори (використано 3 з 4): A – червоний B – зелений C – синій D - зелений

Стан програми після проведення випробувань	Виведення розфарбованого графа, використовуючи жадібний метод, та інформації про послідовність і використані кольори

Таблиця 5.5 - Тестування алгоритмів розфарбування графа (бектрекінг зі степеневою евристикою)

Мета тесту	Перевірити алгоритм розфарбування бектрекінгом зі степеневою евристикою
Початковий стан програми	Відкрите вікно з вибором методу розфарбування графа
Вхідні дані	Вибір методу бектрекінг зі степеневою евристикою для графу з вершинами A, B, C, D, E, F Список ребер: AB, AC, BD, CD, CE, DF, EF Доступні кольори (маскимум 6): червоний, зелений, синій, жовтий, білий, розовий

Схема проведення тесту	Розфарбувати граф використовуючи послідовність вершин. Послідовність визначається степеню вершини (кількість суміжних вершин) за спаданням. Призначити кольори вершинам так, щоб сусідні вершини мали різний колір
Очікуваний результат	Послідовність вершин зі ступенями для розфарбування: C (3), D (3), A (2), B (2), E (2), F (2) Призначені кольори (використано 2 з 6): A – зелений B – червоний C – червоний D – зелений E - зелений F - червоний
Стан програми після проведення випробувань	Виведення розфарбованого графа, використовуючи бектрекінг зі степеневою евристикою

Таблиця 5.6 - Тестування алгоритмів розфарбування графа (бектрекінг з евристикою MRV)

Мета тесту	Перевірити алгоритм розфарбування бектрекінгом з евристикою MRV
Початковий стан програми	Відкрите вікно з вибором методу розфарбування графа
Вхідні дані	Вибір методу бектрекінг з евристикою MRV для графу з вершинами A, B, C, D, E, F Список ребер: AB, AD, BE, CB, DE, EA, FC Доступні кольори (маскимум 6): червоний, зелений, синій, жовтий, білий, розовий
Схема проведення тесту	Розфарбувати граф використовуючи послідовність вершин. Послідовність визначається кількістю доступних кольорів для кожної наступної вершини і формується у процесі розфарбування. Призначити кольори вершинам так, щоб сусідні вершини мали різний колір

Очікуваний результат	Послідовність вершин для розфарбування: A, B, E, D, C, F Призначені кольори (використано 3 з 6): A – червоний B – зелений C – червоний D – зелений E - синій F - зелений
Стан програми після проведення випробувань	Виведення розфарбованого графа, використовуючи бектрекінг з евристикою MRV

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe відкривається вікно з привітанням, коротким описом функціоналу програми і пропозицією створити новий граф (рисунок 6.1).

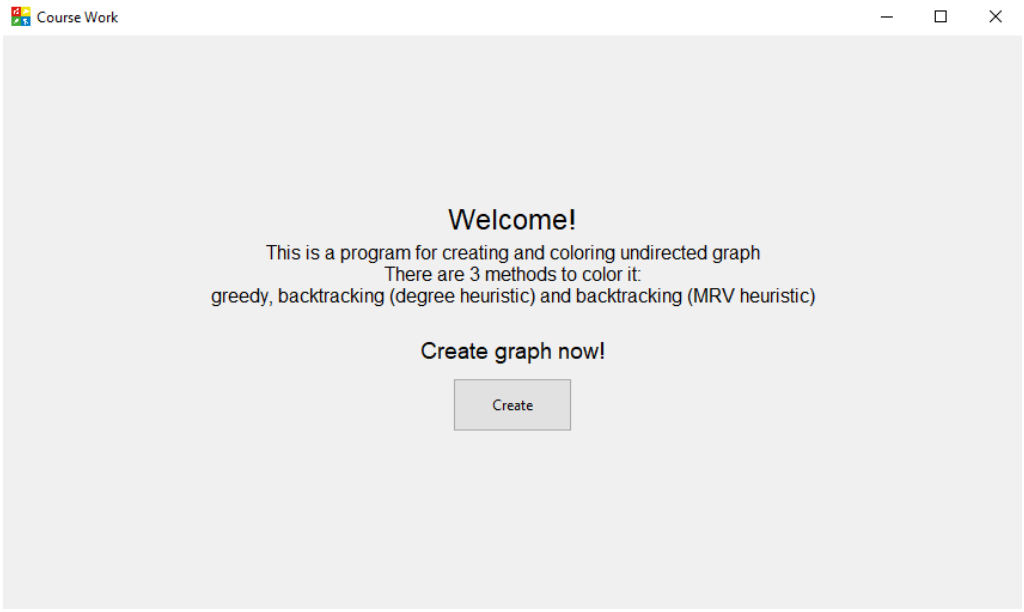


Рисунок 6.1 – Привітальне вікно

Після натиснення кнопки «Create» відкривається сторінка для введення кількості вершин графа (рис. 6.2). Потрібно ввести число від 1 до 20, інакше програма повідомить (рис. 6.3) про неправильно введені дані і попросить ввести їх ще раз.

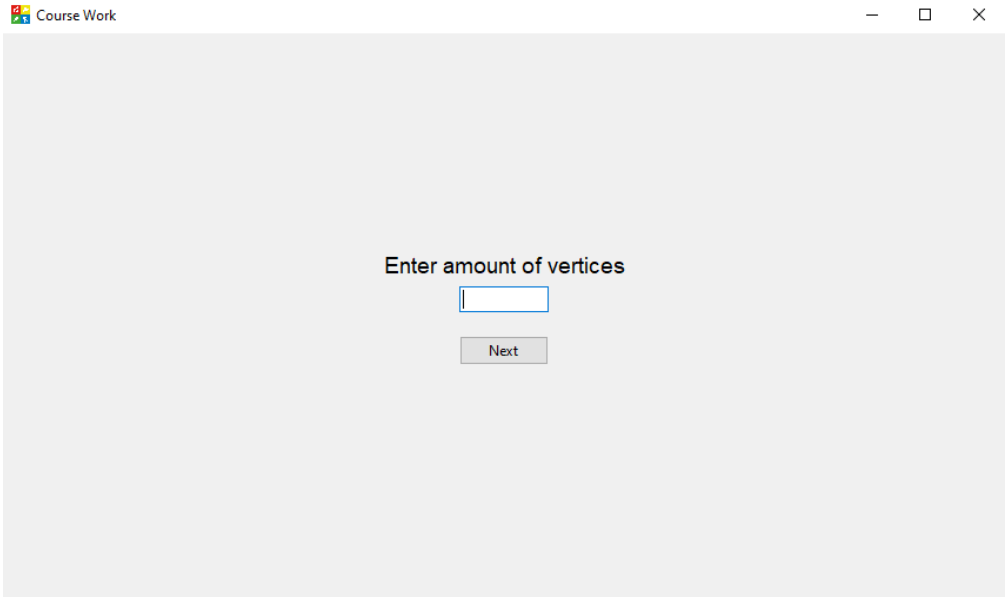
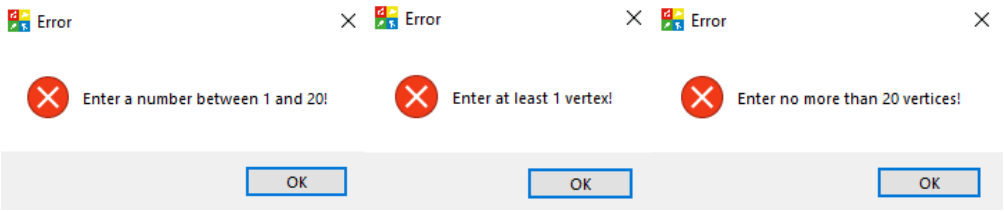


Рисунок 6.2 – Сторінка для введення кількості вершин графа



Рисунки 6.3 – Повідомлення про неправильно введені дані

Далі потрібно обрати чи хочете ви додати ребра до вершини (рис. 6.4), а на наступній сторінці (рис. 6.5) потрібно ввести усі суміжні вершини зі списку, розділені пробілом (вершини поза списком або випадковий набір символів не буде додано до списку ребер, якщо залишити це поле пустим, то жодної вершини не буде додано) і натиснути «Done». Ці операції потрібно провести для кожної вершини графа.

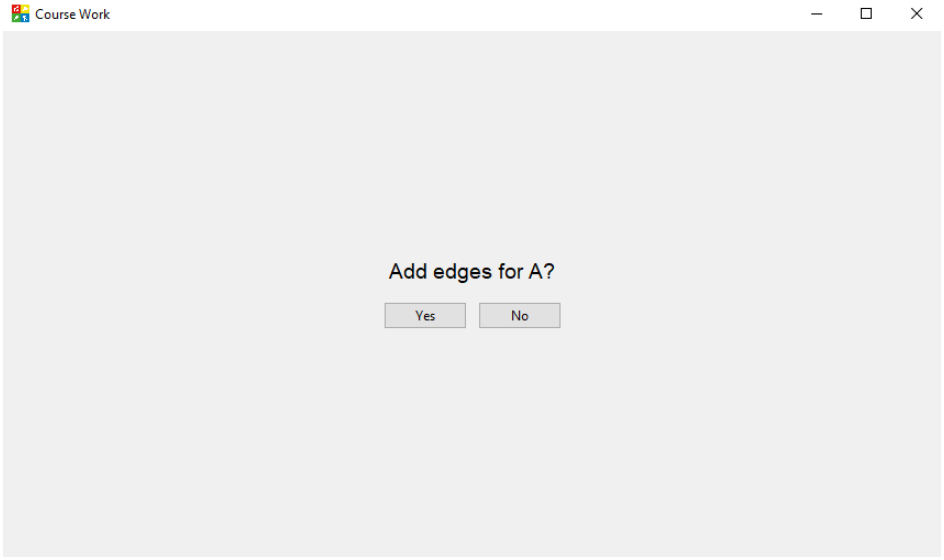


Рисунок 6.4 – Сторінка вибору додавання ребер до вершини

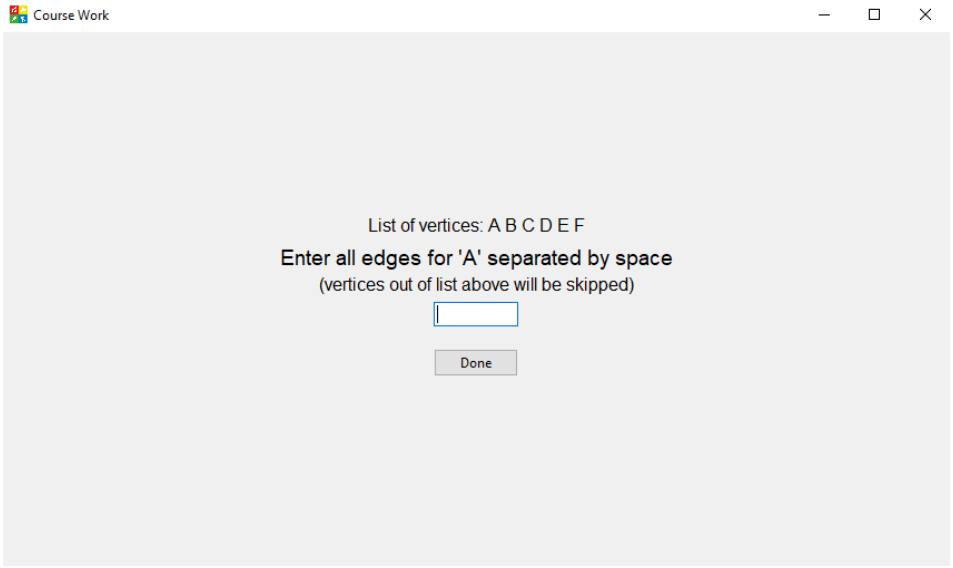


Рисунок 6.5 – Сторінка введення ребер до вершини

На наступній сторінці (рис. 6.6) можна обрати один з наступних варіанті: вивести графічне зображення графа (кнопка «Graph»), перейти до вибору методу розфарбування графа (кнопка «Coloring»), створити новий граф (кнопка «Reset graph») або закрити програму (кнопка «Exit»).

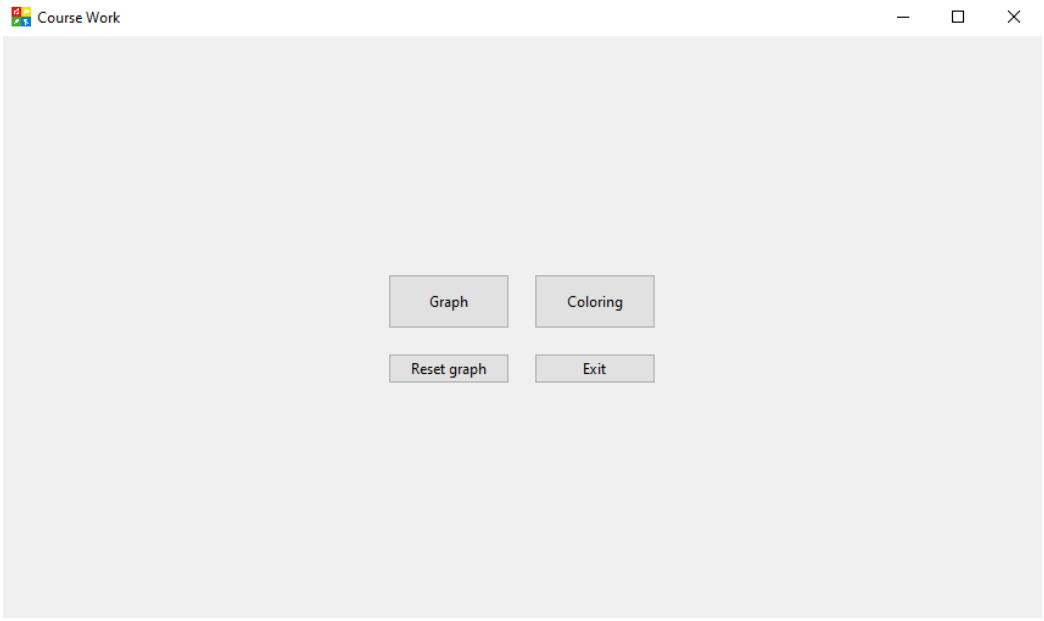


Рисунок 6.6 – Сторінка вибору наступних дій

Якщо граф є пустим, то графічне зображення та розфарбування графа не будуть доступними (рис. 6.7).

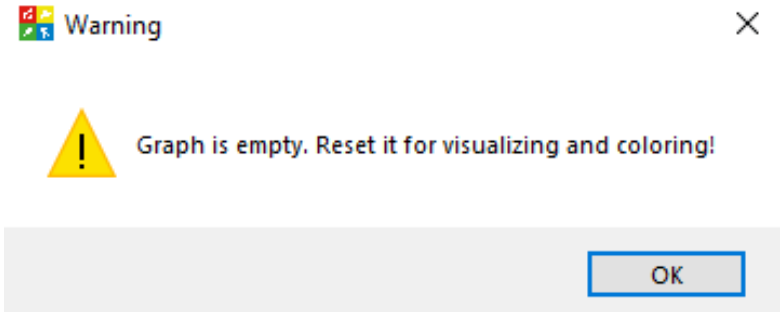


Рисунок 6.7 – Попередження при введені пустого графа

На сторінці (рис. 6.8) вибору методу розфарбування можна обрати один з наступних варіантів: жадібний метод розфарбування (кнопка «Greedy»), бектрегінг зі степеневою евристикою (кнопка «Backtracking (degree)»), бектрекінг з евристикою MRV (кнопка «Backtracking (MRV)»), повернутись до попередньої сторінки (кнопка «Back») або завершити програму (кнопка «Exit»).

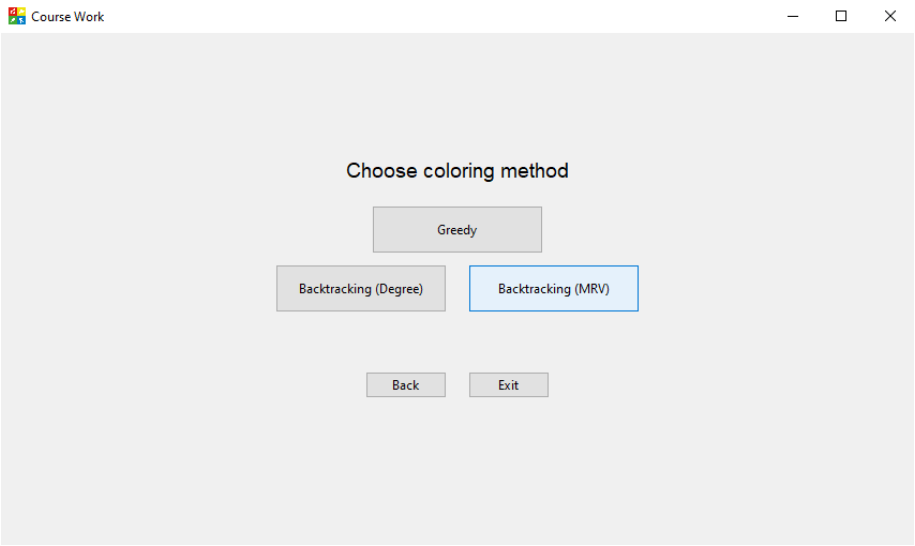


Рисунок 6.8 – Сторінка вибору метода розфарбування

6.2 Формат вхідних та вихідних даних

Користувач задає кількість кількість вершин у графі цілим числом від 1 до 20. Граф будується на основі списку ребер графа. Для кожної вершини окремо вводяться суміжні вершини (якщо ребро додавалося у попередній вершині, то повторне введення не є обов'язковим). Дані про створенний граф записуються до файлу «graph_data.pickle».

Вихідними даними є графічне зображення простого та розфарбованого графа, а також такі дані, як список ізольованих вершин (за наявності), послідовність розфарбування вершин для конкретного методу та кольори, які були для цього використані.

6.3 Системні вимоги

Системні вимоги програмного забезпечення наведено у таблиці 6.1

Таблиця 6.1 – Системні вимоги до програмного забезпечення

	Мінімальні	Рекомендовані
Операційна система	Windows 10 (з останніми обновленнями)	Windows 10 (з останніми обновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2

Продовження таблиці 6.1

	Мінімальні	Рекомендовані
Оперативна пам'ять	2 GB RAM	8 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	800x600	1024x768 або краще
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове ПЗ	TKinter, NetworkX, Matplotlib.pyplot, Pickle	

7 АНАЛІЗ РЕЗУЛЬТАТІВ

Головною задачею даної курсової роботи була реалізація розфарбування графа трьома методами: жадібний метод, метод бектрекінгу (степенева евристика) та метод бектрекінгу (евристика MRV).

Ситуації, при яких програма працює некоректно, не були виявлені, за винятком повідомлень з помилкою або попередженням для користувача за умови неправильних вхідних значень або введення пустого графа. Всі введені користувачем дані програмно перевіряються перед обробкою.

Для перевірки правильності розфарбування неорієнтованого графа буде використано онлайн-пристрій.

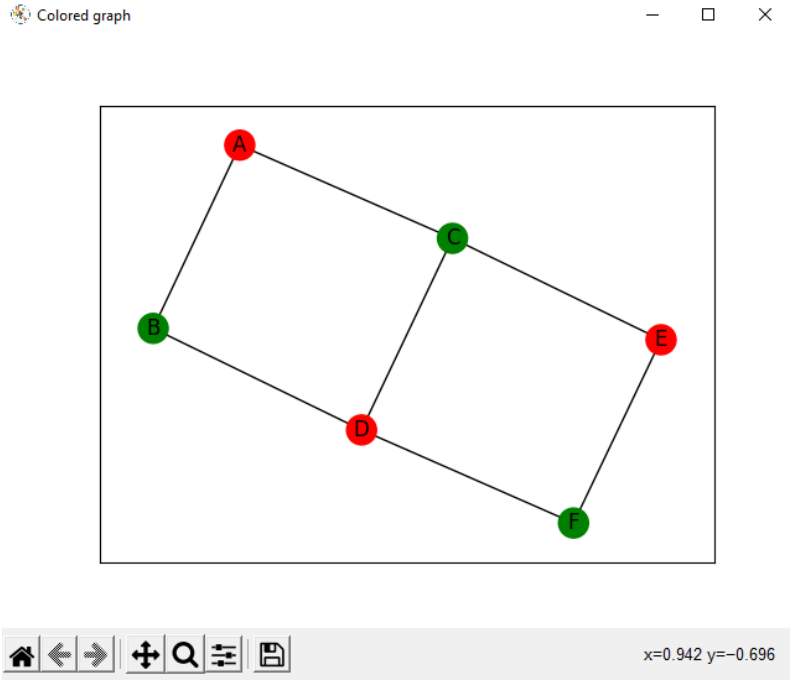


Рисунок 7.1 – Результат роботи програми для розфарбування графа

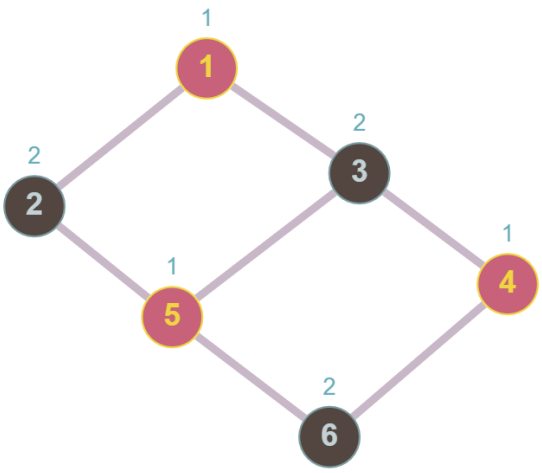


Рисунок 7.2 – Перевірка розфарбування графа в онлайн-пристрої

Таблиця 7.1 – Тестування ефективності методів

Кількість вершин	Параметри тестування	Методи		
		Жадібний	Бектрекінг (степенева евристика)	Бектрекінг (евристика MRV)
5	Кількість ітерацій	25	155	55
10	Кількість ітерацій	100	860	210
25	Кількість ітерацій	625	10025	1275
50	Кількість ітерацій	2500	71250	5050
100	Кількість ітерацій	10000	535000	20100

Візуалізація результатів таблиці 7.1 наведена на рисунку 7.3

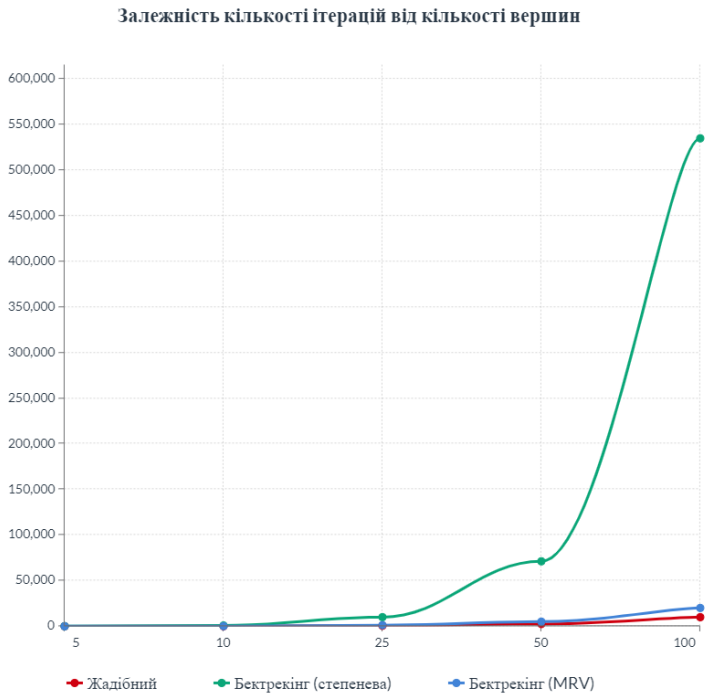


Рисунок 7.3 – Графік залежності кількості ітерацій у методі від кількості вершин графа

За результатами тестування можна зробити наступні висновки:

а) Кількість ітерацій для кожного з протестованих алгоритмів дуже відрізняється і залежить від кількості оброблювальних вершин. Однак кількість ітерацій не змінюється при змінені зв’язків між вершинами графа.

б) Складність алгоритму жадібного методу становить O(n^2). Складність пошуку з поверненням має більшу складність, особливо степенева евристика.

в) Не дивлячись на найкращий показник на графіку залежності, жадібний метод не є найкращим, оскільки не завжди є точним. Пошук з поверненням з використанням степеневої евристики має найгіршу залежність від кількості вершин. Тому найоптимальнішим методом буде пошук з поверненням, використовуючи евристику MRV.

ВИСНОВКИ

В даній курсовій роботі було досліджено побудова і розфарбування неорієнтованого графа трьома методами: жадібним, методом бектрекінгу зі степеневою евристикою та бектрекінгу з евристикою MRV. Виконано реалізацію поставленої задачі з використанням ООП у мові програмування Python за допомогою засобів для реалізації графічного інтерфейсу. Вивчено базові принципи об’єктно-орієнтованого проектування, такі як поліморфізм та інкапсуляція.

У першому розділі розділі була поставлена і описана задача, а саме розроблено декілька підзадач та позначено яким має бути ввід та вивід.

У другому розділі було наведено теоретичні аспекти поставленої задачі, способи задання та виведення графа, розкрито термін розфарбування графа, а також описано його методи.

Опис загального алгоритму та алгоритму підзадач було наведено у третьому розідлі.

Для четвертого розділу було створено UML діаграму класів програмного забезпечення і описано усі використані стандартні та користувацькі методи.

У п’ятому розділі було розроблено план тестування програмного забезпечення, а також змістовні приклади тестування для кожного пункту плану.

У шостому розділі було складено інструкцію користувача, де описано правила роботи з програмою, для більш детального пояснення було додано ілюстрації. У цьому ж розділі описано формат вхідних та вихідних даних і розроблено таблицю із системними вимогами, необхідними для запуску програми.

В останньому розділі було проаналізовано результати тестування, порівняно результат виконання програми зі стороннім онлайн-пристроєм, а також перевірено залежність кількості ітерацій кожного алгоритму від кількості вершин у графі і обрано найоптимальніший метод.

ПЕРЕЛІК ПОСИЛАНЬ

1. Граф (математика)
https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)
2. Розфарбування графів
https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%84%D0%B0%D1%80%D0%B1%D0%BE%D0%B2%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%B2
3. Робота з графами онлайн
https://graphonline.ru/
4. Constraint Satisfaction Problems
https://www.ccs.neu.edu/home/camato/5100/csp.pdf
5. Жадібний алгоритм
https://uk.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D1%96%D0%B1%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра інформатики та програмної інженерії

Затвердив

Керівник Головченко М. М.

«____» _____2022 р.

Виконавець:

Студент Нікулін П. Ю.

«____» _____2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Розфарбування графів»

з дисципліни:

«Основи програмування»

Київ 2022

1. *Мета*: Метою курсової роботи є розробка розробка програмного забезпечення для створення і розфарбування графа трьома методами (жадібним та пошуком з поверненням (бектрекінг) з MRV та степеневою евристиками).
2. *Дата початку роботи*: «__2__» ____липня_____ 2022 р.
3. *Дата закінчення роботи*: «_12_» ____липня_____ 2022 р.
4. *Вимоги до програмного забезпечення*.
    1) Функціональні вимоги:
        - Можливість задання кількості вершин графа
        - Можливість задання суміжностей для кожної вершини графа
        - Можливість збереження побудованого графа у бінарний файл
        - Можливість графічного відображення заданого графа
        - Можливість розфарбування графа різними методами
        - Можливість графічного відображення розфарбованого графа та виведення додаткової інформації
    2) Нефункціональні вимоги:
        - Можливість запуску ПЗ на ОС Windows 10
        - Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:
          ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.
          ГОСТ 19.106 - 78 - Вимоги до програмної документації.
          ГОСТ 7.1 - 84 та ДСТУ 3008 - 95 - Розробка технічної документації.
5. *Стадії та етапи розробки*:
    1) Об'єктно-орієнтований аналіз предметної області задачі (до__.__.202_ р.)
    2) Об'єктно-орієнтоване проектування архітектури програмної

системи (до __.__.202_р.)

3) Розробка програмного забезпечення (до __.__.202_р.)
4) Тестування розробленої програми (до __.__.202_р.)
5) Розробка пояснювальної записки (до __.__.202_ р.)
6) Захист курсової роботи (до __.__.202_ р.).

6. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду «Розфарбування графів»
(Найменування програми (документа))

Електронний носій
(Вид носія даних)

15 рк, 69912 Кб
(Обсяг програми (документа), арк, Кб)

студента 1 курсу групи ІП-14

Нікуліна Павла Юрійовича

main.py

```
from funcs import *

initialization()

root.mainloop()
```

funcs.py

```
from graph_class import *
from root_config import root
from tkinter import ttk, messagebox
import pickle
import os

def initialization(): # create graph
    def start_create(frame):
        frame.destroy()
        create_graph()

    init_frame = ttk.Frame(root.access())
    init_frame.place(anchor='center', relx=0.5, rely=0.5)

    ttk.Label(init_frame, text="Welcome!", font=('Arial', 18)).pack()
    ttk.Label(init_frame, text="This is a program for creating and coloring undirected graph\nThere are 3 methods to color it:\ngreedy, backtracking (degree heuristic) and backtracking (MRV heuristic)", font=('Arial', 12), justify='center').pack()
    ttk.Label(init_frame, text="\nCreate graph now!", font=('Arial', 14)).pack()
    ttk.Button(init_frame, text="Create", command=lambda: start_create(init_frame), width=15,).pack(ipady=10, pady=10)

def close_app():
    if os.path.exists("files/graph_data.pickle"):
```

```
        os.remove("files/graph_data.pickle")

    root.destroy()

def create_graph():
    G = Graph()

    def end_create(): # destroy init page
        with open("files/graph_data.pickle", 'wb') as f: # write graph to binary file
            pickle.dump(G, f)

        output()

    def append_edge(vertex, edges_frame, end_list, i, n): # append edge to graph
        for edge_el in end_list.split():
            if edge_el in G.get_vertex():
                G.add_edge(vertex, edge_el)
                G.set_matrix(i, edge_el)

        if i < n - 1:
            edges_frame.destroy()
            fill_page(i + 1, n)
        else:
            end_create()

    def add_page(vertex, q_frame, i, n): # add edges for vertex
        q_frame.destroy()

        edges_frame = ttk.Frame(root.access())
        edges_frame.place(anchor='center', relx=0.5, rely=0.5)
```

```
        ttk.Label(edges_frame, text=f"List of vertices: {' '.join(G.get_vertex())}",
font=('Arial', 12)).pack(pady=5)
        ttk.Label(edges_frame, text=f"Enter all edges for '{vertex}' separated by space",
font=('Arial', 14)).pack()
        ttk.Label(edges_frame, text="(vertices out of list above will be skipped)",
font=('Arial', 12)).pack()

        end_edge = ttk.Entry(edges_frame, width=10, font=("Arial", 10))
        end_edge.focus_force()
        end_edge.pack(pady=5)

        ttk.Button(edges_frame, text="Done", command=lambda: append_edge(vertex,
edges_frame, end_edge.get(), i, n)).pack(pady=15)

    def skip_vertex(q_frame, i, n): # don't add edges for vertex
        q_frame.destroy()
        if i < n - 1:
            fill_page(i + 1, n)
        else:
            end_create()

    def fill_page(i, n): # fill graph
        q_frame = ttk.Frame(root.access())
        q_frame.place(anchor='center', relx=0.5, rely=0.5)

        ttk.Label(q_frame, text=f"Add edges for {G.get_vertex(i)}?", font=('Arial',
14)).pack(pady=5)

        q_buttons = ttk.Frame(q_frame)
```

```
        q_buttons.pack(pady=10)

        ttk.Button(q_buttons, text='Yes', command=lambda: add_page(G.get_vertex(i),
q_frame, i, n)).grid(row=0, column=0, padx=5)
        ttk.Button(q_buttons, text='No', command=lambda: skip_vertex(q_frame, i,
n)).grid(row=0, column=1, padx=5)

    def launch_func(i): # check conditions and start filling graph
        if n_entry.get().isdigit():
            num = int(n_entry.get())
            n_frame.destroy()

            if 20 >= num > 0:
                G.init_matrix(num)
                for c in range(65, 65 + num): # list of vertices (english alphabet)
                    G.add_vertex(chr(c))

                # for c in range(1, num + 1): # list of vertices (numbers)
                #     G.add_vertex(str(c))

                fill_page(i, num)
            else:
                if num > 20:
                    messagebox.showerror("Error", "Enter no more than 20 vertices!")
                elif num <= 0:
                    messagebox.showerror("Error", "Enter at least 1 vertex!")
                for el in root.winfo_children():
                    el.destroy()
                create_graph()
        else:
```

```
            messagebox.showerror("Error", "Enter a number between 1 and 20!")
            for el in root.winfo_children():
                el.destroy()
            create_graph()

    # start page
    n_frame = ttk.Frame(root.access())
    n_frame.place(anchor='center', relx=0.5, rely=0.5)

    ttk.Label(n_frame, text='Enter amount of vertices', font=('Arial', 14)).pack()
    n_entry = ttk.Entry(n_frame, width=10, font=("Arial", 10))
    n_entry.focus_force()
    n_entry.pack(pady=5)

    ttk.Button(n_frame, text='Next', command=lambda:
launch_func(0)).pack(pady=15)

def output(): # visualize graph or select method to color it
    def reset_graph(frame): # reset graph and create again
        frame.destroy()
        create_graph()

    def output_graph(frame): # visualize graph with info
        if len(G.get_vertex()) > len(G.check_isolated()) > 0:
            frame.destroy()
            ttk.Label(root.access(), text=f"Isolated vertices:\n{'
'.join(G.check_isolated())}", font=('Arial', 14), justify='center').place(anchor='center',
relx=0.5, rely=0.5)
            G.visualize()
        elif len(G.check_isolated()) == len(G.get_vertex()):
```

```
            messagebox.showwarning("Warning", "Graph is empty. Reset it for
visualizing and coloring!")
        else:
            G.visualize()

        output()

    def end_output(frame): # coloring option was chosen
        if len(G.check_isolated()) == len(G.get_vertex()):
            messagebox.showwarning("Warning", "Graph is empty. Reset it for
visualizing and coloring!")
        else:
            frame.destroy()
            coloring_graph()

    for el in root.winfo_children():
        el.destroy()

    with open("files/graph_data.pickle", 'rb') as f: # get graph from binary file
        G = pickle.load(f)

    # output options
    output_frame = ttk.Frame(root.access())
    output_frame.place(anchor='center', relx=0.5, rely=0.5)

    ttk.Button(output_frame, text='Graph', command=lambda:
output_graph(output_frame), width=15).grid(row=0, column=0, padx=10, pady=10,
ipady=10)
    ttk.Button(output_frame, text='Coloring', command=lambda:
end_output(output_frame), width=15).grid(row=0, column=1, padx=10, pady=10,
```

```
ipady=10)

    ttk.Button(output_frame, text='Reset graph', command=lambda:
reset_graph(output_frame), width=15).grid(row=1, column=0, padx=10, pady=10)
    ttk.Button(output_frame, text='Exit', command=close_app, width=15).grid(row=1,
column=1, padx=10, pady=10)

def coloring_graph(): # choose method to color graph
    def greedy_start(): # launch greedy method
        choice_frame.destroy()

        coloring_info = ttk.Label(root.access(), font=("Arial", 14), justify='center') #
greedy method info
        coloring_info.place(anchor='center', relx=0.5, rely=0.5)
        G.greedy(coloring_info)

        coloring_graph()

    def degree_start(): # launch backtracking method with degree heuristic
        choice_frame.destroy()

        coloring_info = ttk.Label(root.access(), font=("Arial", 14), justify='center') #
backtracking (degree) info
        coloring_info.place(anchor='center', relx=0.5, rely=0.5)
        G.backtracking_degree(coloring_info)

        coloring_graph()

    def mrv_start(): # launch backtracking method with MRV heuristic
        choice_frame.destroy()
```

```
        coloring_info = ttk.Label(root.access(), font=("Arial", 14), justify='center') #
backtracking (MRV) info
        coloring_info.place(anchor='center', relx=0.5, rely=0.5)
        G.backtracking_mrv(coloring_info)

        coloring_graph()

    with open("files/graph_data.pickle", 'rb') as f: # get graph from binary file
        G = pickle.load(f)

    for el in root.winfo_children():
        el.destroy()

    # method choosing page
    choice_frame = ttk.Frame(root.access())
    choice_frame.place(anchor='center', relx=0.5, rely=0.5)

    ttk.Label(choice_frame, text="Choose coloring method", font=("Arial", 14),
justify='center').pack()

    choice_buttons = ttk.Frame(choice_frame)
    choice_buttons.pack(pady=20)

    ttk.Button(choice_buttons, text='Greedy', command=greedy_start,
width=25).pack(ipady=10)

    backtracking_buttons = ttk.Frame(choice_buttons)
    backtracking_buttons.pack(pady=10)
    ttk.Button(backtracking_buttons, text='Backtracking (Degree)',
```

```
command=degree_start, width=25).grid(row=0, column=0, padx=10, ipady=10)
    ttk.Button(backtracking_buttons, text='Backtracking (MRV)',
command=mrv_start, width=25).grid(row=0, column=1, padx=10, ipady=10)

    # navigation buttons
    nav_frame = ttk.Frame(choice_frame)
    nav_frame.pack(pady=25)

    ttk.Button(nav_frame, text='Back', command=output).grid(row=0, column=0,
padx=10)
    ttk.Button(nav_frame, text='Exit', command=close_app).grid(row=0, column=1,
padx=10)
```

graph_class.py

```
import networkx as nx
import matplotlib.pyplot as plt

class Graph:
    def __init__(self):
        self.__edges = [] # edges list
        self.__matrix = [] # adjacency matrix
        self.__vertices = [] # list of vertices
        self.__colors = ["red", "green", "blue", "yellow", "white", "fuchsia",
"darkgreen", "cyan", "purple", "gray", "violet", "orange", "lime", "lightblue",
"brown", "olive", "magenta", "coral", "khaki", "wheat"] # list of all colors

    def init_matrix(self, n): # initialization of adjacency matrix
        for i in range(n):
            self.__matrix.append([])
            for j in range(n):
```

```
            self.__matrix[i].append(0)

    def set_matrix(self, row, col_str): # update adjacency matrix
        col = self.__vertices.index(col_str)
        self.__matrix[row][col], self.__matrix[col][row] = 1, 1

    def __vertices_degree(self): # get degree of vertices
        degree = []
        for i in range(len(self.__matrix)):
            degree.append(sum(self.__matrix[i]))

        return degree

    def add_vertex(self, vertex): # add vertex to list
        self.__vertices.append(vertex)

    def get_vertex(self, *args): # get vertex with index or get list of vertices
        if len(args) > 0:
            return self.__vertices[args[0]]
        else:
            return self.__vertices

    def add_edge(self, a, b): # add edge to list of edges
        if [a, b] not in self.__edges and [b, a] not in self.__edges:
            self.__edges.append([a, b])

    def check_isolated(self): # look for isolated vertices
        isolated = []
        for vertex in self.__vertices:
            observer = 0
```

```
            for edge in self.__edges:
                if vertex in edge:
                    observer += 1

            if observer == 0:
                isolated.append(vertex)

        return isolated

    def __color_graph(self, colors, seq): # color graph using sequence
        colors_seq = [''] * len(self.__vertices)
        for vertex in seq:
            set_color = colors[vertex]
            colors_seq[self.__vertices.index(vertex)] = set_color[0]
            curr_edges = self.__matrix[self.__vertices.index(vertex)]
            for i in range(len(curr_edges)):
                if curr_edges[i] == 1 and (set_color[0] in colors[self.__vertices[i]]) and
vertex != self.__vertices[i]:
                    colors[self.__vertices[i]].remove(set_color[0])

        return colors_seq

    def visualize(self, *args): # visualization of graph
        vg = nx.Graph()
        for vertex in self.__vertices:
            vg.add_node(vertex)
        vg.add_edges_from(self.__edges)

        if len(args) > 0:
            nx.draw_networkx(vg, node_color=args[0])
```

```
            plt.gcf().canvas.set_window_title("Colored graph")
        else:
            nx.draw_networkx(vg)
            plt.gcf().canvas.set_window_title("Graph")

        plt.show()

    # coloring methods
    def greedy(self, info): # greedy algorithm
        n = len(self.__matrix)

        # set colors
        g_colors = {}
        for i in range(n):
            g_colors[self.__vertices[i]] = self.__colors[:n]

        greedy_coloring = self.__color_graph(g_colors, self.__vertices) # coloring graph

        # create info for greedy method
        greedy_used = []
        for color in self.__colors[:n]:
            if color in greedy_coloring:
                greedy_used.append(color)

        info.configure(text=f"Sequence of vertices to color (greedy method)\n{' '.join(self.__vertices)}\n\nUsed colors:\n{' '.join(greedy_used)}")

        self.visualize(greedy_coloring) # visualizing graph

    def backtracking_degree(self, info): # backtracking with degree heuristic
```

```
n = len(self.__matrix)

# set colors
bd_colors = {}
for i in range(n):
    bd_colors[self.__vertices[i]] = self.__colors[:n]

# find optimal way to color graph
bd_sequence = []
indices = []
index = 0
for _ in range(n):
    max_degree = -1
    for i in range(n):
        if i not in indices and self.__vertices_degree()[i] > max_degree:
            max_degree = self.__vertices_degree()[i]
            index = i
    indices.append(index)
    bd_sequence.append(self.__vertices[index])

# coloring graph
bd_coloring = self.__color_graph(bd_colors, bd_sequence)

# create info for backtracking (degree) method
bd_used = []
for color in self.__colors[:n]:
    if color in bd_coloring:
        bd_used.append(color)

info.configure(text=f"Sequence of vertices to color (degree backtracking)\n{'
```

```
'.join(bd_sequence)}\n\nUsed colors:\n{' '.join(bd_used)}")

        self.visualize(bd_coloring) # visualizing graph

    def backtracking_mrv(self, info): # backtracking with MRV heuristic
        n = len(self.__matrix)

        # set colors
        mrv_colors = {}
        for i in range(n):
            mrv_colors[self.__vertices[i]] = self.__colors[:n]

        # available colors for each vertex
        mrv_available = {}
        for i in range(n):
            mrv_available[self.__vertices[i]] = n

        # find optimal way to color graph
        mrv_sequence = []
        indices = []
        index = 0

        for _ in range(n):
            min_colors = 21
            for i in range(n):
                if i not in indices and mrv_available[self.__vertices[i]] < min_colors:
                    min_colors = mrv_available[self.__vertices[i]]
                    index = i
            indices.append(index)
            mrv_sequence.append(self.__vertices[index])
```

```
            for edge in self.__edges:
                if self.__vertices[index] in edge and self.__vertices[index] == edge[0] and
edge[1] not in mrv_sequence:
                    mrv_available[edge[1]] -= 1
                elif self.__vertices[index] in edge and self.__vertices[index] == edge[1]
and edge[0] not in mrv_sequence:
                    mrv_available[edge[0]] -= 1

        # coloring graph
        mrv_coloring = self.__color_graph(mrv_colors, mrv_sequence)

        # create info for backtracking (MRV) method
        mrv_used = []
        for color in self.__colors[:n]:
            if color in mrv_coloring:
                mrv_used.append(color)

        info.configure(text=f"Sequence of vertices to color (MRV backtracking)\n{'
'.join(mrv_sequence)}\n\nUsed colors:\n{' '.join(mrv_used)}")

        self.visualize(mrv_coloring) # visualizing graph
```

root_config.py

```
from tkinter import *
class App:
    def __init__(self):
        self.__master = Tk()

        # window resolution
        root_width = 854
```

```
        root_height = 480

        # get the screen dimension
        screen_width = self.__master.winfo_screenwidth()
        screen_height = self.__master.winfo_screenheight()

        # find the center point
        center_x = int(screen_width / 2 - root_width / 2) - 9
        center_y = int(screen_height / 2 - root_height / 2) - 35

        # window settings
        self.__master.title('Course Work')
        self.__master.iconphoto(True, PhotoImage(file='files/kpi_logo.png'))
        self.__master.geometry(f'{root_width}x{root_height}+{center_x}+{center_y}')
        self.__master.resizable(width=True, height=True)

    # access methods
    def mainloop(self):
        return self.__master.mainloop()

    def access(self):
        return self.__master

    def destroy(self):
        return self.__master.destroy()

    def winfo_children(self):
        return self.__master.winfo_children()

root = App() # create window
```