МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

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

КУРСОВА РОБОТА

З основ програмування
(назва дисципліни)

на тему: Розв’язання задачі про вершинне покриття

Студентки 1-го курсу, групи ІП-11

Друзенко Олександри Юріївни

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-11 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Друзенко Олександри Юріївни

(прізвище, ім'я, по батькові)

1. Тема роботи _____

2. Строк здачі студентом закінченої роботи _____

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 _____

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	10.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	11.05.2022	
4.	Розробка сценарію роботи програми	15.05.2022	
6.	Узгодження сценарію роботи програми з керівником	18.05.2022	
5.	Розробка (вибір) алгоритму рішення задачі	20.05.2022	
6.	Узгодження алгоритму з керівником	21.05.2022	
7.	Узгодження з керівником інтерфейсу користувача	22.05.2022	
8.	Розробка програмного забезпечення	23.05.2022	
9.	Налагодження розрахункової частини програми	26.05.2022	
10.	Розробка та налагодження інтерфейсної частини програми	28.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	30.05.2022	
12.	Тестування програми	02.06.2022	
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)

"___" _____20__ р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 56 сторінок, 6 рисунків, 15 таблиць, 2 посилання.

Об’єкт дослідження: задача про вершинне покриття.

Мета роботи: дослідження методів розв’язання задач про вершинне покриття, створення програмного забезпечення для реалізації методів розв’язання.

Вивчено метод об’єктно-орієнтованого програмування для розробки програмного забезпечення. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація жадібного та Approx-Vertex-Cover методів для розв’язання задачі про вершинне покриття.

ГРАФ, ВЕРШИНИ, РЕБРА, ЖАДІБНИЙ МЕТОД, APPROX-VERTEX-COVER МЕТОД.

ЗМІСТ

ВСТУП

Дана робота присвячена розробці програмного забезпечення для розв’язання задачі про вершинне покриття з використанням об’єктно-орієнтованого програмування. Задача полягає в графічному створенні та відображенні графа та розв’язків роботи алгоритмів.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде знаходити вершинне покриття для заданого графа наступними методами:

а) жадібний метод;

б) метод Approx-Vertex-Cover;

Вхідними даними для даної роботи є граф, який задано в графічному вигляді за допомогою вершин і ребер:

$$G = (V, E),$$

де G – граф, V – множина вершин, E – множина ребер[1]. Програмне забезпечення повинно оброобробляти множину вершин та множину ребер графа.

Вихідними даними для даної роботи являється сукупність вершин, що є розв’язком задачі про вершинне покриття даного графа, які виводяться на екран та змінюють колір на графічному відображені графа. Програмне забезпечення повинно видавати розв’язок за умови, що для вхідних даних обраний метод сходиться. Якщо це не так, то програма повинна вивести відповідне повідомлення.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Граф – це дискретний об’єкт, який може бути заданий двома дискретними множинами: множиною точок, які будемо називати вершинами, та множиною ліній, які з’єднують деякі вершини. Лінії будемо називати ребрами.

Граф можна задати наступним чином:

$$G = (V, E) \tag{2.1}$$

Де V – множина вершин, E⊆V×V – множина ребер. Множину вершин графа G позначають V(G), а множину ребер E(G).

Задача про вершинне покриття - NP-повна задача інформатики в області теорії графів. Часто використовується в теорії складності для доведення NP-повноти більш складних задач.

Вершинне покриття для неорієнтованого графа G = (V, E) - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S. Розміром вершинного покриття називається число вершин, що входять у покриття.

2.1 Жадібний метод

Жадібний алгоритм, по черзі включає вершини в мінімальне вершинне покриття і видаляє всі інцидентні включеній вершині ребра, поки ребра не закінчаться.

Жадібний алгоритм вибирає множини керуючись таким правилом: на кожному етапі вибирається множина, що покриває найбільше число ще не покритих елементів.

2.2 Approx-Vertex-Cover метод

Допоки в множині ребер графа E існують ребра, алгоритм вибирає випадкове ребро графа e = (u, v), додає в рішення S обидві вибрані вершини u і v, видаляє з графа всі ребра, які з’єднувались з вершинами u або v[1], [2].

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
Vertexes	Множина вершин
Edges	Множина ребер
Action	Дія активної кнопки
Current	Збереження ідентифікатора об’єкта на полотні, найближчого до вказівника миші
Max_v	Вершина з найбільшою кількістю ребер
Edge	Ребро
Res	Список результату роботи метода

3.1 Загальний алгоритм

1. ПОЧАТОК
2. ПОЧАТОК ЦИКЛУ
3. ПОКИ працює вікно програми:
   - 3.1. ЯКЩО в змінній Action значення «вершина» і користувач натискає на полотно, ТО створити вершину на полотні і додати її до Vertexes.
   - 3.2. ЯКЩО в змінній Action значення «ребро»:
     - 3.2.1. ЯКЩО користувач вибирає на полотні дві раніше не зв’язані ребрами вершини, ТО створити на полотні ребро між вибраними вершинами та додати його до Edges.
     - 3.2.2. ЯКЩО користувач створює ребро з початком і кінцем в одній вершині, АБО користувач створює ребро між вершиною і ребром, АБО користувач створює ребро між ребрами, АБО користувач створює ребро між вже зв’язаними вершинами, ТО вивести на екран повідомлення про помилку.

3.3. ЯКЩО в змінній Action значення «переміщення» і на об’єкт полотна з можливістю переміщення натиснута ЛКМ, ТО в об’єкті Current змінити значення координат на координати вказівника миші.

3.4. ЯКЩО в змінній Action значення «видалення»:

3.4.1. ЯКЩО натиснутий об’єкт Current належить Vertexes, ТО видалити з Edges всі ребра пов’язані з об’єктом Current, та видалити об’єкт Current з Vertexes.

3.4.2. ЯКЩО натиснутий об’єкт Current належить Edges, ТО видалити об’єкт Current з Edges.

3.5. ЯКЩО в змінній Action значення «approx» і натиснута кнопка «вирішити», ТО вивести на екран рішення алгоритму «approx».

3.6. ЯКЩО в змінній Action значення «greedy» і натиснута кнопка «вирішити», ТО вивести на екран рішення алгоритму «greedy».

4. КІНЕЦЬ ЦИКЛУ
5. КІНЕЦЬ

3.2 Алгоритм Approx-Vertex-Cover методу

1. ПОЧАТОК
2. ЯКЩО множина Edges пуста, АБО в множині Vertexes є вершини з кількістю ребер 0, ТО вивести на екран повідомлення про відсутність вхідних даних. ІНАКШЕ:
3. ПОЧАТОК ЦИКЛУ
4. ПОКИ множина Edges не пуста:
   4.1. Вибираємо випадкове ребро Edge з множини Edges.
   4.2. Додаємо вершини ребра Edge до списку Res.
   4.3. Видаляємо з множини Edges всі ребра пов’язані з вершинами ребра Edge.
5. КІНЕЦЬ ЦИКЛУ
6. Виводимо на екран результати методу Res.

7. КІНЕЦЬ

3.3 Алгоритм жадібного (greedy) методу

1. ПОЧАТОК
2. ЯКЩО множини Vertexes і Edges пусті, ТО вивести на екран повідомлення про відсутність вхідних даних. ІНАКШЕ:
3. ПОЧАТОК ЦИКЛУ
4. ПОКИ множини Vertexes і Edges не пусті:
   4.1. Знаходимо вершину з найбільшою кількістю ребер Max_v.
   4.2. Додаємо вершину Max_v до списку Res.
   4.3. Видаляємо Max_v з множини Vertexes.
   4.4. Видаляємо з множини Edges всі ребра пов’язані з вершиною Max_v.
   4.5. ЯКЩО в множині Vertexes є вершини з кількістю ребер 0, ТО видаляємо їх з Vertexes.
5. КІНЕЦЬ ЦИКЛУ
6. Виводимо на екран результати методу Res.
7. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

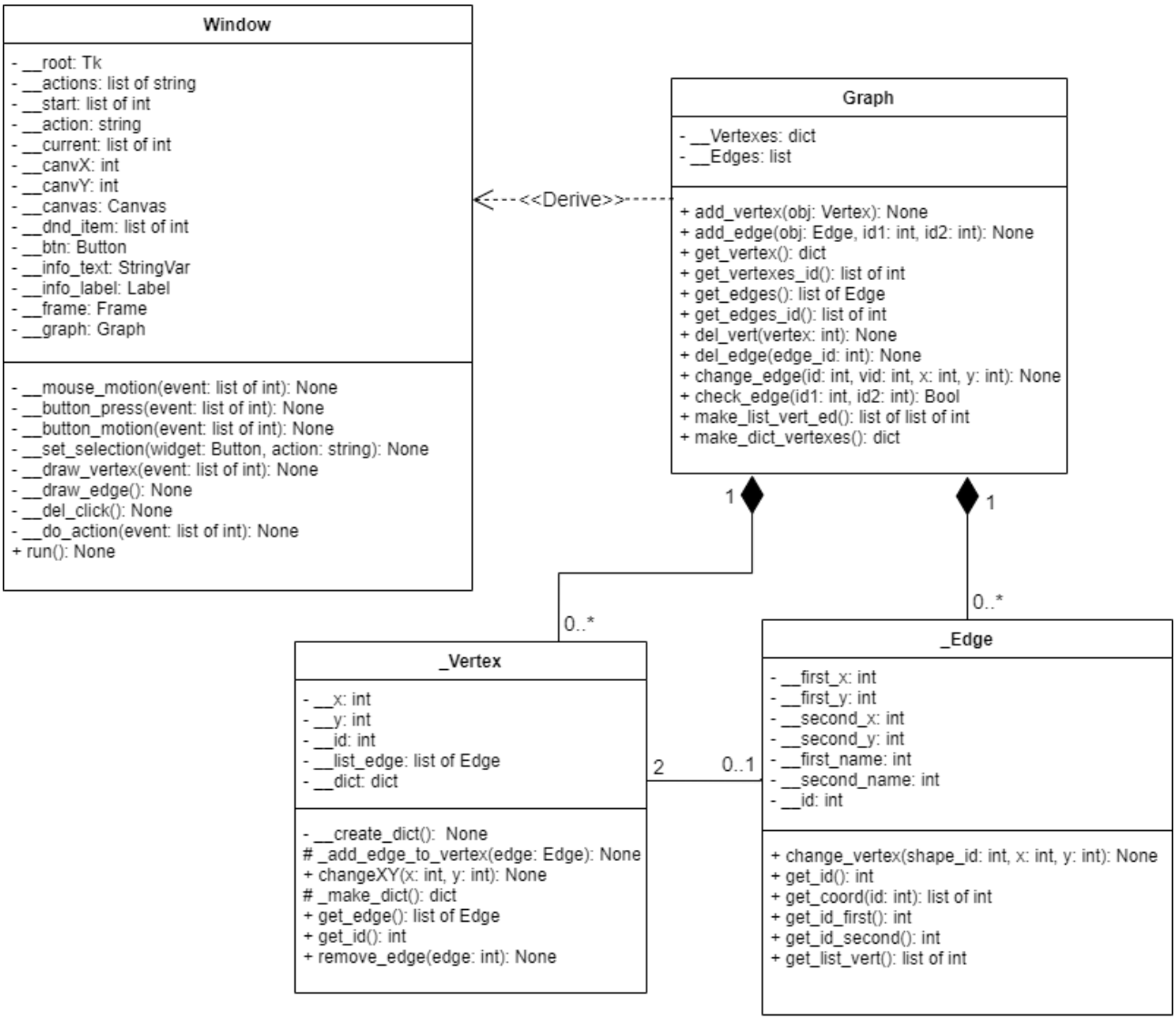

Рисунок 0.1 – Діаграма класів

4.2. Опис методів частин програмного забезпечення

4.2.1. Стандартні методи

У таблиці 4.1 наведено опис стандартних методів, які використовувались при розробці програмного забезпечення.

Таблиця 4.1 – Стандартні методи

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	Tk	title	Встановлення назви віджета	name	-	_tkinter.py
2	Tk	geometry	Встановлення розміру вікна	width x height + x + y	-	_tkinter.py
3	Tk	resizable	Надання дозволу змінення розміру вікна	Bool width, Bool height	-	_tkinter.py
4	Tk	iconbitmap	Встановлення іконки віджета	path string	-	_tkinter.py
5	Tk	configure	Зміна параметрів віджета	background = colour	-	_tkinter.py
6	Tk	Canvas	Створення віджета Canvas	master, bg, width, heigth	-	_tkinter.py
7	Tk	grid	Розміщення віджетів на екрані	row, column, rowspan, columnspan, sticky	-	_tkinter.py
8	Tk	Button	Створення віджета Button	master, text, font, command	-	_tkinter.py

Продовження таблиці 4.1

9	Tk	config	Зміна параметрів віджета	command	-	_tkinter.py
10	Tk	StringVar	Утримувач значень для рядкових змінних	value = string	-	_tkinter.py
11	Tk	Label	Створення віджета мітки	master, relief, textvariable, bg, height, wraplength, justify	-	_tkinter.py
12	Tk	Frame	Створення віджету фрейму	master, background	-	_tkinter.py
13	Canvas	bind	Прив’язування події до віджету	key, callback function	-	_tkinter.py
14	Canvas	tag_bind	Прив’язування події до віджету з тегом	tag, key, callback function	-	_tkinter.py
15	Canvas	find_closest	Знаходження найближчого елементу	x, y	id	_tkinter.py
16	Canvas	find_withtag	Знайти елемент з тегом	tag	id	_tkinter.py
17	Canvas	winfo_width	Повернути висоту віджета	-	width	_tkinter.py

Продовження таблиці 4.1

18	Canvas	winfo_height	Повернути висоту віджета	-	height	_tkinter.py
19	Canvas	coords	Змінити координати	id, x1, y1, x2, y2	-	_tkinter.py
20	Canvas	itemconfig	Зміна параметрів віджета	id, fill	-	_tkinter.py
21	Canvas	move	Переміщення фігури	id, x, y	-	_tkinter.py
22	Canvas	create_line	Створення лінії	x1,y1,x2,y2, fill, width, activefill	-	_tkinter.py
23	Canvas	delete	Видалення фігури	id	-	_tkinter.py
24	Canvas	create_oval	Створення овалу	x1,y1,x2,y2, fill, tags, activefill	-	_tkinter.py
25	Tk	mainloop	Викликати зациклення вікна	-	-	_tkinter.py
26	dict	update	Додати/змінити ключ-значення в словнику	dict	-	builtins.py
27	list	append	Додати об’єкт в кінець списку	obj	-	builtins.py
28	dict	pop	Видалити об’єкт в контейнері	obj	-	builtins.py

Продовження таблиці 4.1

29	list	len	Повернення кількісті елементів у контейнері	list	number	builtins.py

4.2.2. Користувацькі методи

У таблиці 4.2 наведено опис користувацьких методів, які використовувались при розробці програмного забезпечення.

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	Graph	__init__	Конструктор графа	-	-	Graph.py
2	Graph	add_vertex	Додати вершину в множину вершин графа	obj	-	Graph.py
3	Graph	add_edge	Додати ребро в множину ребер графа	obj, id1, id2	-	Graph.py
4	Graph	get_vertexes	Повертає множину вершин графа	-	list of Vertex	Graph.py

Продовження таблиці 4.2

5	Graph	get_vertexes_id	Повертає список ідентифікаторів вершин	-	list of vertex id	Graph.py
6	Graph	get_edges	Повертає множину ребер	-	list of Edge	Graph.py
7	Graph	get_edges_id	Повертає список ідентифікаторів ребер	-	list of edge id	Graph.py
8	Graph	del_vert	Видаляє з множини вершин графа вершину	vertex id	-	Graph.py
9	Graph	del_edge	Видаляє з множини ребер графа ребро	edge id	-	Graph.py
10	Graph	change_edge	Змінює координати ребра	edge id, vertex id, x, y	-	Graph.py
11	Graph	check_edge	Перевірка чи існує ребро між вершинами	vertex id1, id2	Bool	Graph.py
12	Graph	make_list_vert_ ed	Повертає список пар id вершин з’єднаних ребрами	-	list of list of vertex id	Graph.py

Продовження таблиці 4.2

13	Graph	make_dict_vert_ number	Повертає словник вигляду «вершина: кількість ребер»	-	dict	Graph.py
14	_Edge	__init__	Конструктор ребра	item_id, x2, y2, x1, y1, name1, name2	-	Graph.py
15	_Edge	__repr__	Зміна вигляду представлення об’єкту	-	string	Graph.py
16	_Edge	change_end	Змінити координати кінця ребра	shape_id, x, y	-	Graph.py
17	_Edge	get_id	Повертає ідентифікатор ребра	-	edge id	Graph.py
18	_Edge	get_coord	Повертає координати ребра	vertex id	list of int	Graph.py
19	_Edge	get_id_first	Повертає ідентифікатор першої вершини	-	vertex id	Graph.py

Продовження таблиці 4.2

20	_Edge	get_id_second	Повертає ідентифікатор другої вершини	-	vertex id	Graph.py
21	_Edge	get_list_vert	Повертає список вершин	-	list of vertex id	Graph.py
22	_Vertex	__init__	Конструктор вершини	shape, x, y	-	Graph.py
23	_Vertex	__repr__	Зміна вигляду представлення об’єкту	-	string	Graph.py
24	_Vertex	__update_dict	Обновити словник	-	-	Graph.py
25	_Vertex	_add_edge_to_vertex	Додати ребро до вершини	Edge	-	Graph.py
26	_Vertex	changeXY	Змінити координати вершини	x, y	-	Graph.py
27	_Vertex	_make_dict	Повертає словник з вершиною	-	dict	Graph.py
28	_Vertex	get_edge	Повертає список ребер	-	list of Edge	Graph.py
29	_Vertex	get_id	Повертає ідентифікатор вершини	-	id	Graph.py
30	_Vertex	remove_edge	Видаляє ребро зі списку ребр вершини	Edge	-	Graph.py

Продовження таблиці 4.2

31	Window	__init__	Конструктор робочого вікна	width, height, title, resizable, icon	-	Window.py
32	Window	__mouse_motion	Знаходить найближчу до миші фігуру та зафарбовує її	event	-	Window.py
33	Window	__button_press	Зберігає дані натиснутої фігури та викликає переміщення	event	-	Window.py
34	Window	__button_motion	Переміщення об’єкта	event	-	Window.py
35	Window	__set_selection	надає активній кнопці натиснутого вигляду	widget, action	-	Window.py
36	Window	__draw_vertex	Створення вершини	event	-	Window.py
37	Window	__draw_edge	Створення ребра	-	-	Window.py
38	Window	__del_click	Видалення об'єкта	-	-	Window.py

Продовження таблиці 4.2

39	Window	__do_action	Викликає функцію натиснутої кнопки	event	-	Window.py
40	Window	run	Запуск робочого вікна	-	-	Window.py

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 План тестування

Для розробленого програмного забезпечення складемо план тестування, за допомогою якого протестуємо весь основний функціонал та реакцію на виключні ситуації.

а) Тестування правильності створення елементів графу.
   1) Тестування при створенні вершини на вершині.
   2) Тестування при створенні ребра, для вже зв’язаних вершин.
   3) Тестування при створенні ребра з початком в іншому ребрі.
   4) Тестування при створенні ребра з кінцем в іншому ребрі.

б) Тестування коректної роботи функції переміщення елементів графу.
   1) Тестування переміщення вершини з ребрами.
   2) Тестування переміщення ребра.

в) Тестування коректної роботи функції видалення елементів графу.
   1) Тестування видалення вершини з ребрами.
   2) Тестування видалення ребра.

г) Тестування коректності роботи жадібного та approx-cover-vertex методів.
   1) Перевірка роботи методів при введені недостатньої інформації.
   2) Перевірка роботи методів.

Проведемо тестування (таблиці 5.1 – 5.10)

5.2 Приклади тестування

Таблиця 5.1 - Тестування при створенні вершини на вершині

Мета тесту	Перевірити коректність вводу графічних даних
Початковий стан програми	На полотні наявна одна вершина
Вхідні дані	Створити на наявній вершині іншу вершину
Схема проведення тесту	У вікні програми натиснути кнопку «вершина», потім клацнути ЛКМ по наявній вершині
Очікуваний результат	Нова вершина не створиться
Стан програми після проведення випробувань	На полотні наявна одна вершина, нова не створилась

Таблиця 5.2 - Тестування при створенні ребра, для вже зв’язаних вершин

Мета тесту	Перевірити коректність створення ребер
Початковий стан програми	На полотні наявні дві вершини, з’єднані одним ребром
Вхідні дані	Створення ребра між вершинами
Схема проведення тесту	У вікні програми натиснути кнопку «ребро», клацнути ЛКМ по першій вершині, потім по другій
Очікуваний результат	Виведення на екран вікна застереження, ребро не створено
Стан програми після проведення випробувань	На екран вивелось повідомлення про неможливість данної дії, ребро не створено

Таблиця 5.3 - Тестування при створенні ребра з початком в іншому ребрі

Мета тесту	Перевірити коректність створення ребер
Початковий стан програми	На полотні наявні дві вершини, з’єднані одним ребром, та одна не з’єднана вершина
Вхідні дані	Створення ребра між ребром і вершиною
Схема проведення тесту	У вікні програми натиснути кнопку «ребро», клацнути ЛКМ по ребру, а потім по нез’єднаній вершині
Очікуваний результат	Виведення на екран вікна застереження, ребро не створено
Стан програми після проведення випробувань	На екран вивелось повідомлення про неможливість данної дії, ребро не створено

Таблиця 5.4 - Тестування при створенні ребра з кінцем в іншому ребрі

Мета тесту	Перевірити коректність створення ребер
Початковий стан програми	На полотні наявні дві вершини, з’єднані одним ребром, та одна не з’єднана вершина
Вхідні дані	Створення ребра між вершиною і ребром
Схема проведення тесту	У вікні програми натиснути кнопку «ребро», клацнути ЛКМ по нез’єднаній вершині, а потім по ребру

Продовження таблиці 5.4

Очікуваний результат	Виведення на екран вікна застереження, ребро не створено
Стан програми після проведення випробувань	На екран вивелось повідомлення про неможливість данної дії, ребро не створено

Таблиця 5.5 - Тестування переміщення вершини з ребрами

Мета тесту	Перевірити коректність роботи функції переміщення
Початковий стан програми	На полотні наявні три вершини, з’єднані ребрами
Вхідні дані	Переміщення вершини
Схема проведення тесту	У вікні програми натиснути кнопку «переміщення», клацнути ЛКМ по вершині, і перемістити її утримуючи ЛКМ
Очікуваний результат	Вершина разом зі з’єднаними кінцями ребер переміститься
Стан програми після проведення випробувань	Вершина разом зі з’єднаними кінцями ребер перемістилася

Таблиця 5.6 - Тестування переміщення ребра

Мета тесту	Перевірити коректність роботи функції переміщення
Початковий стан програми	На полотні наявні три вершини, з’єднані ребрами
Вхідні дані	Переміщення ребра

Продовження таблиці 5.6

Схема проведення тесту	У вікні програми натиснути кнопку «переміщення», клацнути ЛКМ по ребру, і перемістити його утримуючи ЛКМ
Очікуваний результат	Ребро не переміститься
Стан програми після проведення випробувань	Ребро не перемістилося

Таблиця 5.7 - Тестування видалення вершини з ребрами

Мета тесту	Перевірити коректність роботи функції видалення
Початковий стан програми	На полотні наявні три вершини, з’єднані ребрами
Вхідні дані	Видалити вершину
Схема проведення тесту	У вікні програми натиснути кнопку «видалення», клацнути ЛКМ по вершині
Очікуваний результат	На полотні залишиться дві вершини і одне ребро між ними
Стан програми після проведення випробувань	На полотні залишилося два з’єднаних ребра

Таблиця 5.8 - Тестування видалення ребра

Мета тесту	Перевірити коректність роботи функції видалення
Початковий стан програми	На полотні наявні дві вершини, з’єднані ребром
Вхідні дані	Видалити ребро

Продовження таблиці 5.8

Схема проведення тесту	У вікні програми натиснути кнопку «видалення», клацнути ЛКМ по ребру
Очікуваний результат	На полотні залишиться дві не з’єднані вершини
Стан програми після проведення випробувань	На полотні залишилося дві не з’єднані вершини

Таблиця 5.9 - Перевірка роботи методів при введені недостатньої інформації

Мета тесту	Перевірити методи на можливість виведення результатів при відсутності достатніх вхідних даних
Початковий стан програми	Пусте полотно
Вхідні дані	-
Схема проведення тесту	У вікні програми вибрати метод, та натиснути кнопку «вирішити»
Очікуваний результат	На екрані з’явиться повідомлення про недостатню кількість вхідних даних
Стан програми після проведення випробувань	На екрані з’явилося повідомлення про недостатню кількість вхідних даних

Таблиця 5.10 - Перевірка роботи методів

Мета тесту	Перевірити методи на можливість виведення результатів при наявності достатніх вхідних даних
Початковий стан програми	Пусте полотно
Вхідні дані	Граф з трьох з’єднаних вершин
Схема проведення тесту	У вікні програми вибрати метод, та натиснути кнопку «вирішити»

Продовження таблиці 5.10

Очікуваний результат	На екрані з’явиться інформація про результати роботи метода
Стан програми після проведення випробувань	На екрані з’явилася інформація про результати роботи метода

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe, відкривається головне вікно програми (Рисунок 6.1).

Рисунок 6.1 – Головне вікно програми

Далі за допомогою нижнього блоку кнопок, які зображені на рисунку 6.2 відбувається робота з графом.

Рисунок 6.2 – Нижній блок кнопок

Натиснувши кнопку «вершина», користувач може створювати вершини. Щоб створити вершину, необхідно на полотні натиснути ЛКМ.

Натиснувши кнопку «ребро», користувач може створювати ребра. Щоб створити ребро між двома вершинами, потрібно на кожну вершину клацнути ЛКМ один раз.

Натиснувши кнопку «переміщення», користувач може переміщувати вершини по полотну.

Натиснувши кнопку «видалення», користувач може видаляти з графа вершини та ребра.

Далі за допомогою правої частини вікна програми, зображеної на рисунку 6.3, користувач вирішує задачу вершинного покриття.

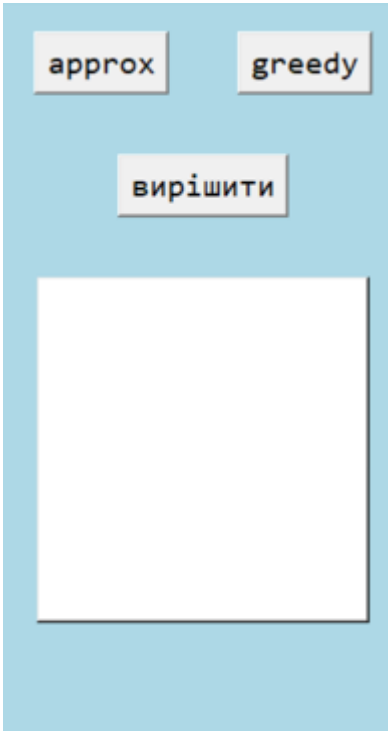

Рисунок 6.3 – Частина вікна роботи з методами

Натиснувши кнопки «approx» або «greedy», користувач обирає метод розв’язання задачі вершинного покриття для побудованого графа.

Щоб розв’язати задачу, потрібно натиснути кнопку «вирішити». Тоді у білому вікні під кнопкою вирішити, з’явиться текст з обраними вершинами та їх кількістю. А на полотні графа вибрані вершини стануть синіми. В залежності від вибраного метода, ребра графа можуть повністю стати синіми («greedy»), або ж тільки ті, які вибрав алгоритм «approx» для вирішення задачі.

6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подається граф у графічному вигляді.

Результатом виконання програми є розв’язок задачі вершинного покриття даного графа, який видається у графічному та тестовому вигляді.

6.3 Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Операційна система	Windows 10 (з останніми обновленнями)	Windows 10/Windows 11 (з останніми обновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
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	1024x768	1024x768 або краще
Прилади введення	Клавіатура, комп’ютерна миша	Прилади введення
Додаткове програмне забезпечення	Python 3.7 або вище	

7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для розв’язання задачі вершинного покриття графа наступними методами: жадібний метод, approx-vertex-cover.

Критичні ситуації у роботі програми виявлені не були. Під час тестування помилок не було виявлено.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористалася обраховуванням та аналізом результатів вручну. Все зійшлось.

Зробимо тестування ефективності алгоритмів розв’язання задачі вершинного покриття графа, та наведемо результати в таблиці 7.1:

Таблиця 7.1 – Тестування ефективності методів

Розмірність системи	Параметри тестування	Метод	
		жадібний	approx-vertex-cover
8 вершин 7 ребер	Кількість ітерацій	4	3
	Кількість елементарних операцій	144	65
	Кількість вершин	4	6
8 вершин 13 ребер	Кількість ітерацій	4	3
	Кількість елементарних операцій	213	105
	Кількість вершин	4	6
15 вершин 22 ребра	Кількість ітерацій	8	7
	Кількість елементарних операцій	575	327
	Кількість вершин	8	14
25 вершин 40 ребер	Кількість ітерацій	13	12
	Кількість елементарних операцій	1475	926
	Кількість вершин	13	24

Візуалізація результатів таблиці 7.1 наведено на рисунках 7.1 та 7.2 (червоний - approx-vertex-cover, зелений – жадібний):

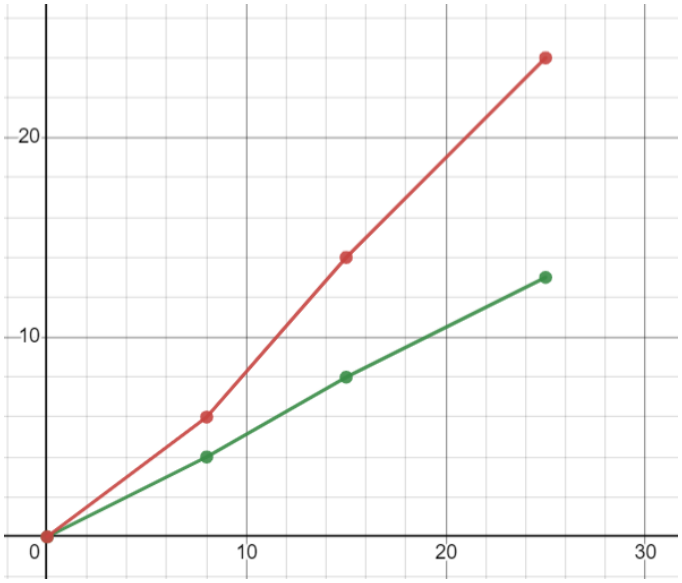

Рисунок 7.1 – Графік залежності кількості вибраних вершин від кількості вершин

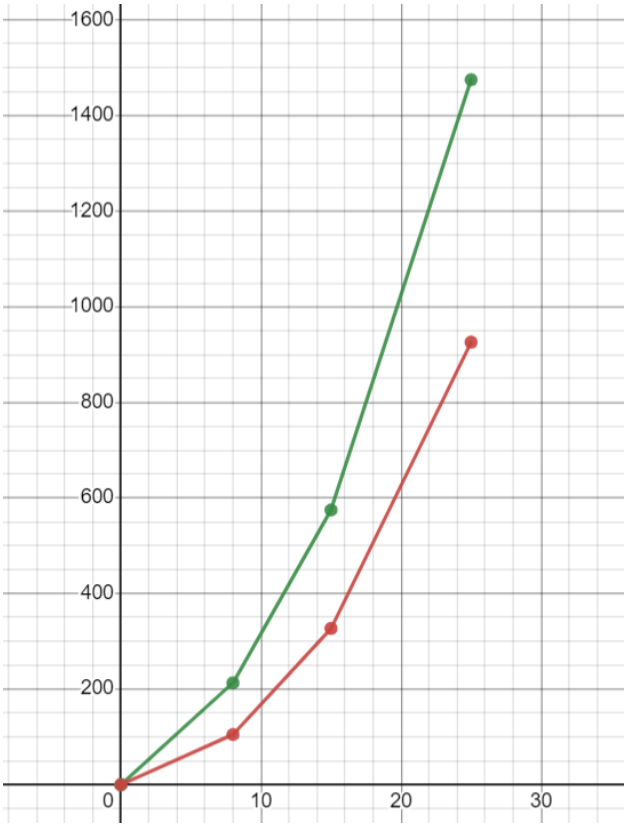

Рисунок 7.2 – графік залежності кількості елементарних кроків від кількості вершин

За результатами тестування можна зробити такі висновки:

а) Всі розглянуті методи дозволяють знаходити розв’язки задачі про вершинне покриття графа.

б) Складність всіх розглянутих методів є квадратичною, і приблизно дорівнює – $O(k * n^2)$, де k – кількість ітерацій виконаних методом, n – кількість вершин.

в) З розглянутих методів найоптимальнішим для практичного використання є метод approx-vertex-cover, оскільки він виконується найшвидше. Але якщо потрібен результат, а не швидкість, то потрібно використовувати жадібний метод.

ВИСНОВОК

Отже, ми дослідили методи та створили алгоритми для розв’язання задачі про вершинне покриття, створили програмне забезпечення, за допомогою якого можна створити граф та використовуючи методи, знайти розв’язок задачі. В розробці програмного забезпечення використовували об’єктно-орієнтоване програмування. Протестували та проаналізували нашу програму.

ПЕРЕЛІК ПОСИЛАНЬ

1. Головченко М. М. Алгоритми та структури даних: курс лекцій. Київ : КПІ, 2021. 226 с.
2. Задача про покриття множини. Вікіпедія URL: https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BF%D1%80%D0%BE_%D0%BF%D0%BE%D0%BA%D1%80%D0%B8%D1%82%D1%82%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%B8

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко Максим Миколайович

«___»_____201_ р.

Виконавець:

Студентка Друзенко Олександра Юріївна

«10» ___травня____2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: розв’язання задачі про вершинне покриття

з дисципліни:

«Основи програмування»

Київ 2022

1. *Мета*: Метою курсової роботи є розробка програмного забезпечення для розв’язання задачі вершинного покриття.
2. *Дата початку роботи*: «10»__травня____2022 р.
3. *Дата закінчення роботи*: «12»_червня__ 2022 р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      – Можливість графічного створення та зображення графа.
      – Можливість переміщення вершин по області поля для графа.
      – Можливість видалення вершин та ребр у графі.
      – Можливість розв’язання задачі вершинного покриття за допомогою жадібного метода та Approx-Vertex-Cover метода.
      – Можливість графічного виведення результатів роботи методів.
   2) Нефункціональні вимоги:
      – Операційна система Windows 10
      – Роздільна здатність екрану більше чим 1100 x 500
      – 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 - 2015 - 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 програмного забезпечення вирішення задачі вершинного покриття

(Найменування програми (документа))

Github

(Вид носія даних)

17 арк, 58 Кб

(Обсяг програми (документа), арк., Кб)

студента групи ІП-11 І курсу
Друзенко О.Ю.

main.py

```
import sys
from Window import Window

window = Window(1075, 495, 'Вершинне покриття', resizable=(False, False), icon=f'{sys.path[0]}\ico\graph2.ico')
window.run()
```

Window.py

```
import tkinter as tk
from tkinter import Tk
from functools import partial
from Graph import Graph
from tkinter import messagebox
from Metods import *
import sys

class Window:
    def __init__(self, width, height, title='My window', resizable=(False, False), icon=None):
        self.__root = Tk()
        self.__root.title(title)
        self.__root.geometry(f'{width}x{height}+200+200')
        self.__root.resizable(resizable[0], resizable[1])
        if icon:
            self.__root.iconbitmap(icon)
        self.__root.configure(background='light blue')
        self.__actions = ('вершина', 'ребро', 'переміщення', 'видалення', 'approx', 'greedy')
        self.__start = None  # координати
```

```
        self.__action = None
        self.__current = None
        self.__canvX = 800
        self.__canvY = 450
        self.__canvas = tk.Canvas(self.__root, bg="white", width=self.__canvX,
height=self.__canvY)
        self.__canvas.grid(row=0, columnspan=4)
        self.__dnd_item = ()

        for i in range(4):
            __btn = tk.Button(self.__root, text=self.__actions[i], font=('Consolas', 16))
            __btn.config(command=partial(self.__set_selection, __btn, self.__actions[i]))
            __btn.grid(column=i, row=1, sticky=tk.W + tk.E + tk.S + tk.N)
        self.__frame = tk.Frame(background='light blue')
        for i in range(4, 6):
            __btn = tk.Button(self.__frame, text=self.__actions[i], font=('Consolas', 16))
            __btn.config(command=partial(self.__set_selection, __btn, self.__actions[i]))
            __btn.grid(column=i - 4, row=0, sticky=tk.W + tk.E + tk.S + tk.N, pady=20,
padx=23)
        __btn = tk.Button(self.__frame, text='вирішити', font=('Consolas', 16),
command=lambda:
            self.__do_action(0) if (self.__action == 'approx' or self.__action == 'greedy')
else None)
        __btn.grid(columnspan=2, column=0, row=1, padx=35, pady=20)

        self.__info_text = tk.StringVar(value='')
        self.__info_label = tk.Label(self.__frame, textvariable=self.__info_text,
bg='white', height=15, relief=tk.RAISED,
                                     wraplength=200, justify=tk.LEFT)
        self.__info_label.grid(columnspan=2, column=0, row=2, padx=25, pady=20,
sticky=tk.W + tk.E)
```

```
        self.__frame.grid(columnspan=2, column=5, rowspan=3, row=0, sticky=tk.W + tk.S + tk.N)
        self.__canvas.bind("<Motion>", self.__mouse_motion)
        self.__canvas.bind("<Button-1>", self.__do_action)
        self.__canvas.tag_bind("draggable", "<ButtonPress-1>", self.__button_press)
        self.__graph = Graph()

    def __mouse_motion(self, event):  # зафарбовує в жовтий ближню фігуру до вказівника
        if self.__action != "greedy" and self.__action != 'approx':
            self.__canvas.itemconfig(self.__current, fill="red")
            self.__current = self.__canvas.find_closest(event.x, event.y)
            self.__canvas.itemconfig(self.__current, fill="yellow")

    def __button_press(self, event):      # зберігає данні теперішнього об'єкта та викликає функцію його переміщення
        item = self.__canvas.find_withtag(tk.CURRENT)
        self.__dnd_item = (item, event.x, event.y)
        self.__canvas.tag_bind("draggable", "<Button1-Motion>", self.__button_motion)

    def __button_motion(self, event):  # функція переміщення об'єкта
        x, y = event.x, event.y
        try:
            vertex_id, x0, y0 = self.__dnd_item
        except TypeError:
            return
        if x - 15 <= 0 or y - 15 <= 0 or x + 15 >= self.__canvas.winfo_width() or y + 15 >= self.__canvas.winfo_height():
            if x - 5 <= 0:
```

```
            x = 15
        elif x + 5 >= self.__canvas.winfo_width():
            x = self.__canvX - 15
        elif y - 5 <= 0:
            y = 15
        elif y + 5 >= self.__canvas.winfo_height():
            y = self.__canvY - 15
    else:
        self.__canvas.move(vertex_id, x - x0, y - y0)
        for ed in self.__graph.get_edges():
            x, y = event.x, event.y
            if ed.get_id_first() == vertex_id[0]:
                coord = ed.get_coord(vertex_id[0])
                self.__canvas.coords(ed.get_id(), x, y, *coord)
                self.__graph.change_edge(ed.get_id(), vertex_id[0], x, y)
            elif ed.get_id_second() == vertex_id[0]:
                coord = ed.get_coord(vertex_id[0])
                self.__canvas.coords(ed.get_id(), *coord, x, y)
                self.__graph.change_edge(ed.get_id(), vertex_id[0], x, y)

        self.__dnd_item = (vertex_id, x, y)
        self.__graph.get_vertexes()[vertex_id[0]][0].changeXY(x, y)     # словник вершин[id вершини][0] = об'єкт вершина
        self.__graph.add_vertex(self.__graph.get_vertexes()[vertex_id[0]][0])

    def __set_selection(self, widget, action):     # функція надання активній кнопці натиснутого вигляду
        if self.__frame in widget.master.winfo_children():  # якщо рамка знаходиться на одному рівні з віджетом
            for w in widget.master.winfo_children():
                w.config(relief=tk.RAISED)
```

```
            for w in self.__frame.winfo_children():
                w.config(relief=tk.RAISED)
            widget.config(relief=tk.SUNKEN)
        else:
            for w in self.__frame.master.winfo_children():
                w.config(relief=tk.RAISED)
            for w in widget.master.winfo_children():
                w.config(relief=tk.RAISED)
            widget.config(relief=tk.SUNKEN)
        self.__canvas.itemconfig(tk.ALL, fill="red")
        self.__action = action  # передаємо змінній текст вибраної кнопки


    def __draw_vertex(self, event):  # функція створення вершини
        x, y = event.x, event.y
        if self.__current:
            if self.__current[0] in self.__graph.get_vertexes_id():
                cur_x, cur_y = self.__graph.get_vertexes()[self.__current[0]][1:3]
                if abs(cur_x - x) >= 35 or abs(cur_y - y) >= 35:
                    self.__graph.add_vertex(Graph._Vertex(self.__canvas.create_oval(x -
15, y - 15, x + 15, y + 15,
                                                                    fill="red",    activefill="yellow",
tags=("draggable")), x, y))
            else:
                self.__graph.add_vertex(Graph._Vertex(self.__canvas.create_oval(x - 15, y
- 15, x + 15, y + 15, fill="red",
                                                                activefill="yellow",
tags=("draggable")), x, y))
        else:
            self.__graph.add_vertex(Graph._Vertex(self.__canvas.create_oval(x - 15, y -
15, x + 15, y + 15, fill="red",
```

```
                                                activefill="yellow",
tags=("draggable")), x, y))

    def __draw_edge(self):    # функція створення ребра
        if self.__current:
            if not self.__start:
                if self.__current[0] in self.__graph.get_vertexes_id():
                    cv = self.__graph.get_vertexes()[self.__current[0]]
                    self.name1 = self.__current[0]
                    self.__start = [cv[1], cv[2]]
                if self.__current[0] in self.__graph.get_edges_id():
                    tk.messagebox.showinfo('ребро', "не можна проводити ребра із ребер")
                    return
            else:
                flag = 0
                if self.__current[0] in self.__graph.get_edges_id():
                    flag = tk.messagebox.showinfo('ребро', "не можна проводити ребро в
ребро")
                elif self.name1 == self.__current[0]:
                    flag = tk.messagebox.showinfo('ребро', "ребро виходить та входить в
одну вершину")
                elif self.__graph.check_edge(self.name1, self.__current[0]):
                    flag = tk.messagebox.showinfo('ребро', "ребро вже існує")
                if flag:
                    self.__start = None
                    self.name1 = ''
                    return
                else:
                    x2, y2 = self.__start
                    self.__start = None
                    cv = self.__graph.get_vertexes()[self.__current[0]]
```

```
                bbox = (cv[1], cv[2], x2, y2)
                edge = Graph._Edge(self.__canvas.create_line(*bbox, fill="red",
activefill="yellow", width=2),
                                   *bbox, self.name1, self.__current[0])
                self.__graph.add_edge(edge, self.name1, self.__current[0])

    def __del_click(self):   # функція видалення об'єкта
        for elem in self.__graph.get_vertexes().values():  # якщо елемент належить
вершинам
            if self.__current[0] == elem[0].get_id():
                edges = elem[0].get_edge()
                for i in range(len(edges) - 1, -1, -1):
                    if edges[i].get_id_first() == elem[0].get_id():

self.__graph.get_vertexes()[edges[i].get_id_second()][0].remove_edge(
                            edges[i].get_id())  # з об'єкта "вершина" видаляємо ребро
                    elif edges[i].get_id_second() == elem[0].get_id():

self.__graph.get_vertexes()[edges[i].get_id_first()][0].remove_edge(edges[i].get_id()
)
                    self.__canvas.delete(edges[i].get_id())
                    self.__graph.del_edge(edges[i].get_id())
                self.__graph.del_vert(elem[0].get_id())

        for elem in self.__graph.get_edges():  # якщо елемент належить ребрам
            if self.__current[0] == elem.get_id():

self.__graph.get_vertexes()[elem.get_id_first()][0].remove_edge(elem.get_id())

self.__graph.get_vertexes()[elem.get_id_second()][0].remove_edge(elem.get_id())
                self.__graph.del_edge(elem.get_id())
```

```
        self.__canvas.delete(self.__current[0])

    def __do_action(self, event):    # функція яка викликає задані функції
        self.__dnd_item = None
        if self.__action == "вершина":
            self.__draw_vertex(event)
        elif self.__action == "ребро":
            self.__draw_edge()
        elif self.__action == "переміщення":
            self.__button_press(event)
        elif self.__action == "видалення":
            self.__del_click()
        elif self.__action == 'greedy':
            self.__info_text = tk.StringVar(value=greedy(self.__graph, self.__canvas))
        elif self.__action == 'approx':
            self.__info_text = tk.StringVar(value=approx(self.__graph, self.__canvas))

        self.__info_label.config(textvariable=self.__info_text)

    def run(self):
        self.__root.mainloop()


if __name__ == '__main__':
    window = Window(1075, 495, 'Вершинне покриття', resizable=(False, False),
icon=f'{sys.path[0]}\ico\graph2.ico')
    window.run()
```

Graph.py

```
class Graph:
```

```
def __init__(self):
    """Конструктор графа"""
    self.__Vertexes = {}
    self.__Edges = []

def add_vertex(self, obj):
    """Додати вершину в множину вершин графа"""
    self.__Vertexes.update(obj._make_dict())

def add_edge(self, obj, id1, id2):
    """Додати ребро в множину ребер графа"""
    self.__Edges.append(obj)
    self.__Vertexes[id1][0]._add_edge_to_vertex(obj)
    self.__Vertexes[id2][0]._add_edge_to_vertex(obj)

def get_vertexes(self):
    """Повертає множину вершин графа"""
    return self.__Vertexes

def get_vertexes_id(self):
    """Повертає список ідентифікаторів вершин"""
    res = []
    for vertex in self.__Vertexes:
        res.append(vertex)
    return res

def get_edges(self):
    """Повертає множину ребер"""
    return self.__Edges

def get_edges_id(self):
```

```
    """Повертає список ідентифікаторів ребер
    :return:список id ребер"""
    res = []
    for edge in self.__Edges:
        res.append(edge.get_id())
    return res

def del_vert(self, vertex):
    """Видаляє з множини вершин графа вершину
    :param vertex:id вершини
    :type vertex:int"""
    cdict = self.__Vertexes.copy()
    for i in self.__Vertexes:
        if vertex == i:
            cdict.pop(vertex)
    self.__Vertexes = cdict

def del_edge(self, edge_id):
    """Видаляє з множини ребер графа ребро"""
    for i in self.__Edges:
        if edge_id == i.get_id():
            self.__Edges.remove(i)

def change_edge(self, id, vid, x, y):
    for edge in self.__Edges:
        if edge.get_id() == id:
            edge.change_end(vid, x, y)

def check_edge(self, id1, id2):
    res = []
    for edge in self.__Edges:
```

```
            el = edge.get_list_vert()
            if [el[0], el[1]] not in res and [el[1], el[0]] not in res:
                res.append(el)
        if [id1, id2] in res or [id2, id1] in res:
            return True

    def make_list_vert_ed(self):
        res = []
        for edge in self.__Edges:
            el = edge.get_list_vert()
            if [el[0], el[1]] not in res and [el[1], el[0]] not in res:
                res.append(el)
        return res

    def make_dict_vert_number(self):
        res = {}
        for vertex in self.__Vertexes:
            key = vertex
            value = len(self.__Vertexes[vertex][0].get_edge())
            res.update({key: value})
        return res

    class _Edge:
        def __init__(self, item_id, x2, y2, x1, y1, name1, name2):
            self.__first_x = x1
            self.__first_y = y1
            self.__second_x = x2
            self.__second_y = y2
            self.__first_name = name1
            self.__second_name = name2
            self.__id = item_id
```

```
def __repr__(self):
    return f'{self.__id}[{self.__first_name},{self.__second_name}]'

def change_end(self, shape_id, x, y):
    if shape_id == self.__first_name:
        self.__first_x = x
        self.__first_y = y
    elif shape_id == self.__second_name:
        self.__second_x = x
        self.__second_y = y

def get_id(self):
    return self.__id

def get_coord(self, id):
    if id == self.__first_name:
        return [self.__second_x, self.__second_y]
    elif id == self.__second_name:
        return [self.__first_x, self.__first_y]
    elif id == 0:
        return [self.__first_x, self.__first_y, self.__second_x, self.__second_y]

def get_id_first(self):
    return self.__first_name

def get_id_second(self):
    return self.__second_name

def get_list_vert(self):
    return [self.__first_name, self.__second_name]
```

```
class _Vertex:
    def __init__(self, shape, x, y):
        self.__x = x
        self.__y = y
        self.__id = shape
        self.__list_edge = []
        self.__dict = {}
        self.__update_dict()

    def __repr__(self):
        return f'{self.__id}({self.__x},{self.__y})[{self.__list_edge}];'

    def __update_dict(self):
        self.__dict.update({self.__id: [self, self.__x, self.__y, self.__list_edge]})

    def _add_edge_to_vertex(self, edge):
        self.__list_edge.append(edge)

    def changeXY(self, x, y):
        self.__x = x
        self.__y = y

    def _make_dict(self):
        self.__update_dict()
        return self.__dict

    def get_edge(self):
        return self.__list_edge

    def get_id(self):
```

```
        return self.__id

    def remove_edge(self, edge):
        for i in self.__list_edge:
            if edge == i.get_id():
                self.__list_edge.remove(i)
```

Metods.py

```
from random import randint
from tkinter import ALL

def greedy(graph, canvas):
    canvas.itemconfig(ALL, fill="red")
    res = []
    list_edges = graph.make_list_vert_ed()
    dict_vert = graph.make_dict_vert_number()
    if not list_edges and not dict_vert:
        return 'для виконання методу потрібні вершини'
    while list_edges or dict_vert != {}:
        max_value = 0
        max_key = 0
        for key in dict_vert:
            if max_value <= dict_vert[key]:
                max_value = dict_vert[key]
                max_key = key
        dict_vert.pop(max_key)
        res.append(max_key)
        canvas.itemconfig(max_key, fill='blue')

        for edge in graph.get_edges():
```

```
        edges = edge.get_list_vert()
        if max_key in edges :
            canvas.itemconfig(edge.get_id(), fill="blue")

        if max_key in edges:
            if max_key == edges[0] and edges[1] in dict_vert:
                if dict_vert[edges[1]]-1 == 0:
                    dict_vert.pop(edges[1])
                else:
                    dict_vert.update({edges[1]: dict_vert[edges[1]]-1})
            elif max_key == edges[1] and edges[0] in dict_vert:
                if dict_vert[edges[0]]-1 == 0:
                    dict_vert.pop(edges[0])
                else:
                    dict_vert.update({edges[0]: dict_vert[edges[0]]-1})

    copy_list = list_edges.copy()
    for edge in list_edges:
        if max_key in edge:
            copy_list.remove(edge)
    list_edges = copy_list.copy()
    copy_dict = dict_vert.copy()
    for vertex in dict_vert:
        if dict_vert[vertex] == 0:
            canvas.itemconfig(vertex, fill="blue")
            res.append(vertex)
            copy_dict.pop(vertex)
    dict_vert = copy_dict.copy()
return f"результат роботи методу {res}\nкількість вершин:{len(res)}"
```

```
def approx(graph, canvas):
    canvas.itemconfig(ALL, fill="red")
    if graph.get_edges() == []:
        return 'для виконання цього методу потрібні ребра'
    for key in graph.make_dict_vert_number():
        if graph.make_dict_vert_number()[key] == 0:
            return "не всі вершини з'єднані ребрами"


    res = []
    list_edges = graph.make_list_vert_ed()
    while list_edges:
        rand = randint(0, len(list_edges)-1)
        del_item = list_edges.pop(rand)
        res.append(del_item)
        copy_list = list_edges.copy()
        for edge in graph.get_edges():
            if del_item[0] in edge.get_list_vert() and del_item[1] in edge.get_list_vert():
                canvas.itemconfig(edge.get_id(), fill="blue")
        for vertex in del_item:
            canvas.itemconfig(vertex, fill='blue')
            for edge in list_edges:
                if vertex in edge:
                    copy_list.remove(edge)
        list_edges = copy_list.copy()
    return f"результат роботи методу {res}\nкількість вершин:{len(res)*2}"
```