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: 57 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**

[Анотація 4](#_Toc105934631)

[ВСТУП 6](#_Toc105934632)

[**1** **Постановка задачі** 7](#_Toc105934633)

[**2** **Теоретичні відомості** 8](#_Toc105934634)

[2.1 Жадібний метод 8](#_Toc105934635)

[2.2 Approx-Vertex-Cover метод 8](#_Toc105934636)

[**3** **Опис алгоритмів** 9](#_Toc105934637)

[3.1 Загальний алгоритм 9](#_Toc105934638)

[3.2 Алгоритм Approx-Vertex-Cover методу 10](#_Toc105934639)

[3.3 Алгоритм жадібного (greedy) методу 11](#_Toc105934640)

[**4** **Опис програмного забезпечення** 12](#_Toc105934641)

[4.1. Діаграма класів програмного забезпечення 12](#_Toc105934642)

[4.2. Опис методів частин програмного забезпечення 12](#_Toc105934643)

[4.2.1. Стандартні методи 12](#_Toc105934644)

[4.2.2. Користувацькі методи 16](#_Toc105934645)

[**5** **Тестування програмного забезпечення** 22](#_Toc105934646)

[5.1 План тестування 22](#_Toc105934647)

[5.2 Приклади тестування 23](#_Toc105934648)

[**6** **Інструкція користувача** 29](#_Toc105934649)

[6.1 Робота з програмою 29](#_Toc105934650)

[6.2 Формат вхідних та вихідних даних 30](#_Toc105934651)

[6.3 Системні вимоги 30](#_Toc105934652)

[**7** **Аналіз і узагальнення результатів** 32](#_Toc105934653)

[**ВИСНОВОК** 35](#_Toc105934654)

[**ПЕРЕЛІК ПОСИЛАНЬ** 36](#_Toc105934655)

[**Додаток А Технічне завдання** 37](#_Toc105934656)

[**Додаток Б Тексти програмного коду** 40](#_Toc105934657)

# 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:

,

де – граф, – множина вершин, – множина ребер[1].Програмне забезпечення повинно обробробляти 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.

1. **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:

|  |
| --- |
|  |

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 Е існують ребра, алгоритм вибирає випадкове ребро графа е = (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].

1. **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:
   1. ЯКЩО в змінній Action значення «вершина» і користувач натискає на полотно, ТО створити вершину на полотні і додати її до Vertexes.
   2. ЯКЩО в змінній Action значення «ребро»:
      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.
      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. 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.
   4. ЯКЩО в змінній Action значення «видалення»:
      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.
      2. ЯКЩО натиснутий об’єкт Current належить Edges, ТО видалити об’єкт Current з Edges.
   5. ЯКЩО в змінній Action значення «approx» і натиснута кнопка «вирішити», ТО вивести на екран рішення алгоритму «approx».
   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:
   1. Вибираємо випадкове ребро Edge з множини Edges.
   2. Додаємо вершини ребра Edge до списку Res.
   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:
   1. Знаходимо вершину з найбільшою кількістю ребер Max\_v.
   2. Додаємо вершину Max\_v до списку Res.
   3. Видаляємо Max\_v з множини Vertexes.
   4. Видаляємо з множини Edges всі ребра пов’язані з вершиною Max\_v.
   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
8. **Опис програмного забезпечення**
   1. Діаграма класів програмного забезпечення

Діаграма класів розробленого програмного забезпечення наведена на рисунку 4.1.

Diagram

Description automatically generated

Рисунок 4.1 – Діаграма класів

* 1. Опис методів частин програмного забезпечення
     1. Стандартні методи

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

Таблиця 4.1 – Стандартніметоди

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № 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 |

* + 1. 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 |

1. **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.

1. 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.
2. Тестування коректної роботи функції переміщення елементів графу.
   1. Тестування переміщення вершини з ребрами.
   2. Тестування переміщення ребра.
3. Тестування коректної роботи функції видалення елементів графу.
   1. Тестування видалення вершини з ребрами.
   2. Тестування видалення ребра.
4. 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 |

1. **Інструкція користувача**

6.1 Робота з програмою

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

Graphical user interface, application

Description automatically generated

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.

Square

Description automatically generated

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® ІІІ  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 | 1024х768 | 1024х768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | Прилади введення |
| Додаткове програмне забезпечення | Python 3.7 або вище | |

1. **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):

Chart, line chart

Description automatically generated

Рисунок 7.1 – Графік залежності кількості вибраних вершин від кількості вершин

Chart, line chart

Description automatically generated

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 – , де – кількість ітерацій виконаних методом, – 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 : веб-сайт. 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.

1. Нефункціональні вимоги:

* Операційна система Windows 10
* Роздільна здатність екрану більше чим 1100 х 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.

* 1. *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.).
   1. *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**

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі вершинного покриття*

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

*17 арк, 58 Кб*

(Вид носія даних)

*Github*

*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:словник з вершинами {id:obj, x, y, list of Edge}"""

return self.\_\_Vertexes

def get\_vertexes\_id(self):

"""Повертає список ідентифікаторів вершин

:return:список id вершин"""

res = []

for vertex in self.\_\_Vertexes:

res.append(vertex)

return res

def get\_edges(self):

"""Повертає множину ребер

:return:список об'єктів класу Edge"""

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):

"""змінити координати кінця ребра"""

for edge in self.\_\_Edges:

if edge.get\_id() == id:

edge.change\_end(vid, x, y)

def check\_edge(self, id1, id2):

"""перевіряє чи наявне ребро в графі із заданими ребрами"""

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):

"""Створює список ребер, з елементами виду [id 1ї вершини, id 2ї вершини]"""

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):

"""Створює словник вершин виду {id вершини: кількість ребер}"""

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):

"""змінити координати одного з кінців ребра"""

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):

"""Повертає ідентифікатор ребра"""

return self.\_\_id

def get\_coord(self, id):

"""Повертає список з координатами"""

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):

"""Повертає ідентифікатор першої вершини ребра"""

return self.\_\_first\_name

def get\_id\_second(self):

"""Повертає ідентифікатор другої вершини ребра"""

return self.\_\_second\_name

def get\_list\_vert(self):

"""Повертає список ідентифікаторів вершин ребра"""

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):

"""Створення словника для вершини"""

self.\_\_dict.update({self.\_\_id: [self, self.\_\_x, self.\_\_y, self.\_\_list\_edge]})

def \_add\_edge\_to\_vertex(self, edge):

"""Додати до списку ребер вершини нове ребро"""

self.\_\_list\_edge.append(edge)

def changeXY(self, x, y):

"""змінити координати вершини"""

self.\_\_x = x

self.\_\_y = y

def \_make\_dict(self):

"""створити та повернути словник вершини"""

self.\_\_update\_dict()

return self.\_\_dict

def get\_edge(self):

"""повернути список ребер вершини"""

return self.\_\_list\_edge

def get\_id(self):

"""Повернути ідентифікатор вершини"""

return self.\_\_id

def remove\_edge(self, edge):

"""Видалити ребро із вершини"""

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):

"""жадібний метод"""

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: # пошук вершини з найбільшою кількістю ребер

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: #видаляємо ребра пов'язані з вибраною вершиною

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):

"""approx-vertex-cover метод"""

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) #вибираємо випадкове ребро

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: #видаляємо ребра, які прилягають до вибраного ребра

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}"