НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра автоматизованих систем обробки інформації та управління

Спеціальність 122 «Комп'ютерні науки та інформаційні технології»

ЗВІТ

виконання самостійної роботи студентів

з дисципліни: «Технології розробки програмного забезпечення»

на тему: Задача дихотомічного розбиття графа з метою мінімізації сумарної ваги розрізу

Кількість балів: _____

Дата захисту: _____

Члени проекту _____ Как С.Р.

(підпис) (прізвище та ініціали)

_____ Штик В.Л.

(підпис) (прізвище та ініціали)

Перевірила _____ Сперкач М.О.

(підпис) (прізвище та ініціали)

Київ – 2018

ЗМІСТ

ВСТУП

У даній курсовій роботі розглянуто задачу дихотомічного розбиття графа з метою мінімізації сумарної ваги розрізу. Розв'язання даної задачі застосовується зокрема при проектуванні топології локальної мережі трасуванні зв'язків друкованих плат або мікросхем та при розробці графів алгоритмів. Зменшуючи зв'язки між компонентами, розв'язок даної задачі сприяє оптимальному розподілу ресурсів, що в свою чергу максимізує прибутки підприємства або ефективність пристрою [1].

Поставлену задачу можна розв'язати методом повного перебору, генетичним методом, методом бджолиного рою, застосуванням алгоритму Федуччі-Маттеуса або Кернігана-Ліна та багатьма іншими методами [2]. Однак лише метод повного перебору може дати точне рішення. У рамках даної роботи буде розглянуто метод бджолиного рою та алгоритм Федуччі-Маттеуса.

Під час практичного використання алгоритмів можуть виникнути певні проблеми, серед яких – проблема забезпечення приблизної рівності потужностей компонент. Ми використовуємо градаційний критерій, який дозволяє зберегти баланс кількостей вершин у компонентах .

Проблема коректності вхідного графу також є актуальною. Тому для програми реалізовано генератор графів, що гарантує виконання умов зв'язності, відсутності висячих вершин та петель.

Ще однією проблемою є швидкодія. При вирішенні задачі розмірності тисяча, програмі необхідна достатньо велика кількість часу – близько години для розбиття на компоненти.

В реальних задачах може виникнути необхідність розділяти граф більше ніж на дві компоненти. Реалізовані в даній курсовій роботі алгоритми здійснюють дихотомічне розбиття графу. Для розбиття більше ніж на дві компоненти можна застосовувати дані алгоритми на отриманих компонентах допоки не буде досягнуто бажаної кількості компонент.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Задано неорієнтований, зважений граф *G(V,E,w),* де *V* - множина вершин, *E* - множина ребер, *w* - множина ваг ребер. Передбачається, що ваги ребер є цілими числами, граф відповідає умовам зв'язності, не містить петель та висячих вершин.

Ціль задачі полягає в тому, щоб розбити заданий граф на два підграфи таким чином, щоб сумарна вага розрізу набувала мінімального значення і при цьому отримані компоненти були зв’язними. Схожа постановка задачі розглянута у [3]. Для забезпечення зв’язності компонент при локальних покращеннях розв’язку розглядаються лише ті вершини, які є інцидентними ребрам розрізу.

Поставлену задачу цілочисельного лінійного програмування можна розв’язати методом повного перебору, генетичним методом, методом бджолиного рою, застосуванням алгоритму Федуччі-Маттеуса або Кернігана-Ліна та багатьма іншими методами. Однак лише метод повного перебору може дати точне рішення. У рамках даної роботи буде розглянуто метод бджолиного рою та алгоритм Федуччі-Маттеуса.

Розв’язання даної задачі застосовується зокрема при проектуванні топології локальної мережі трасуванні зв’язків друкованих плат або мікросхем та при розробці графів алгоритмів. Зменшуючи зв’язки між компонентами, розв’язок даної задачі сприяє оптимальному розподілу ресурсів, що в свою чергу максимізує прибутки підприємства або ефективність пристрою.

1.2 Опис функціональної моделі

Наведемо опис функцій системи у вигляді діаграми використання (Use Case) на рисунку 1.1:

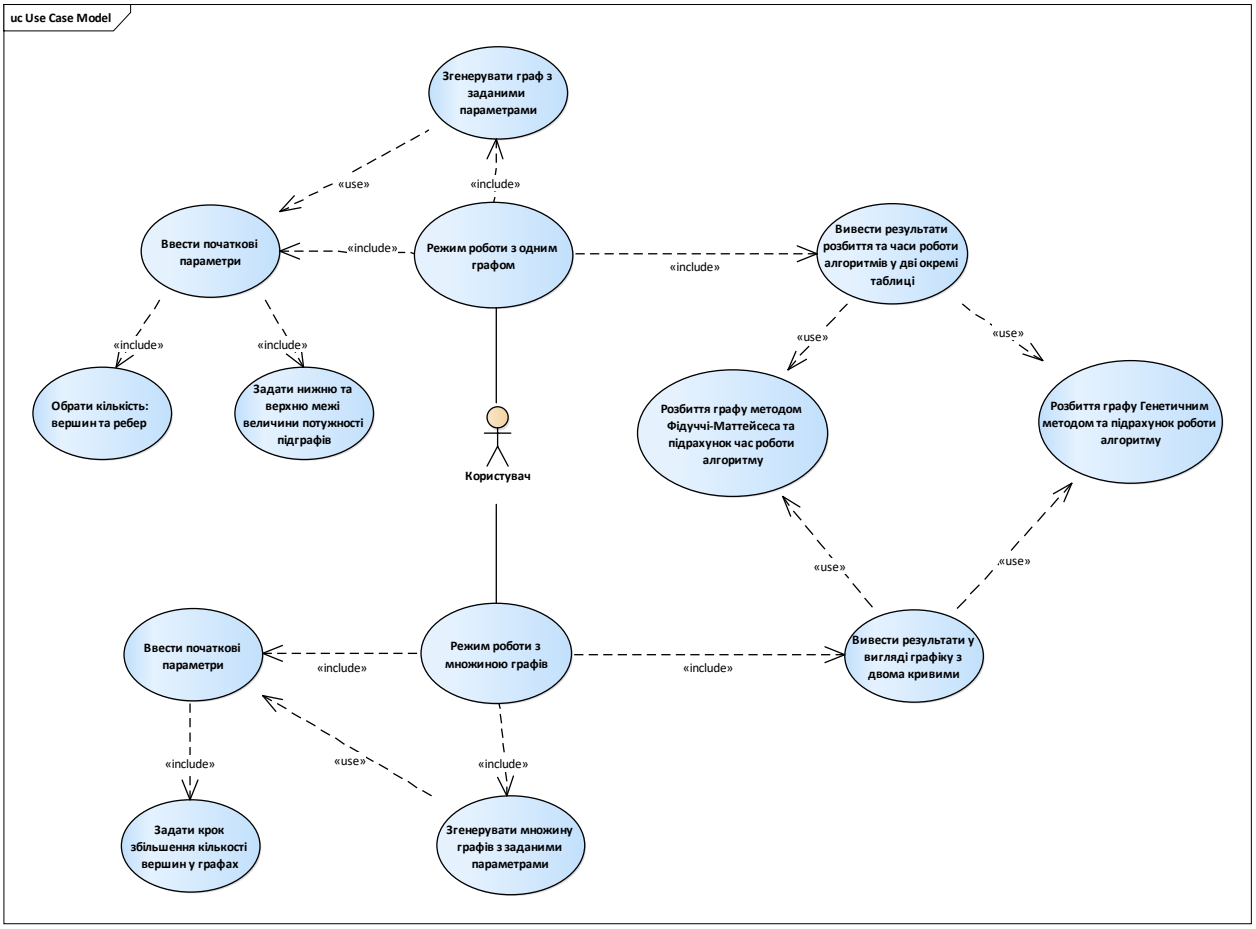

Рисунок 1.1 ─ Use Case діаграма

Наведемо формулювання всіх функціональних вимог у відповідності з діаграмою використання (таблиця 1.1).

Таблиця 1.1 – Функціональні вимоги

Актор	**Варіант використання**	**Функціональна вимога**	**Пріоритет**
Користувач	**Розбиття одного графу**	1. Система надає можливість користувачу розбити граф за допомогою двох методів.	Високий
		1.1. Система надає можливість користувачу ввести параметри графа	Високий
		1.1.1. Система надає можливість користувачу ввести кількість вершин.	Високий
		1.1.2. Система надає можливість користувачу ввести кількість ребер.	Високий
		1.2. Система надає можливість згенерувати граф з вказаними параметрами.	Високий
		1.3. Система надає можливість розбити граф на підграфи.	Високий
		1.3.1. Система надає можливість виконати розбиття графу методом Фідуччі-Маттейсеса та переглянути час роботи алгоритму.	Високий
		1.3.2. Система надає можливість виконати розбиття графу бджолиним методом та переглянути час роботи алгоритму.	Високий

Продовження таблиці 1.1

		1.4. Система надає можливість користувачу ввести необхідні для бджолиного алгоритму параметри .	Високий
	Порівняльний аналіз розбиття множини графів	2. Система надає можливість користувачу порівняти час розбиття множини графів за допомогою двох методів.	Високий
		2.1. Система надає можливість користувачу згенерувати множину графів з заданим параметром.	Високий
		2.1.1. Система надає можливість користувачу ввести крок збільшення кількості вершин у графах.	Високий
		2.2. Система надає можливість вивести результати у вигляді графіку з двома кривими.	Високий
		2.2.1. Система надає можливість виконати розбиття графів методом Фідуччі-Маттейсеса та побудувати криву залежності часу роботи алгоритму від кількості вершин у графі.	Високий
		2.2.2. Система надає можливість виконати розбиття графів бджолиним методом та побудувати гістограму залежності часу роботи алгоритму від кількості вершин у графі.	Високий

2 РІШЕННЯ З ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

2.1.1 Розбиття графа-приклада

Опис вхідних даних:

- **ns** — кількість бджіл-розвідників, ціле число в межах від 4 до 40;
- **mb** — кількість зон концентраціїї розв'язків, ціле число в межах від 2 до кількісті бджіл-розвідників;
- **nf** — кількість бджіл-фуражирів, ціле число в межах від кількості зон концентрації розв’язків помноженої на два до кількості бджіл-розвідників помноженої на п’ять;
- **r —** "Різність" бджіл фуражирів r (кількість змін у бджолах-фуражирах), ціле число в межах від 1 до чверті кількості вершин у графі;
- **stopCount** — допустима кількість ітерацій без зміни значення ЦФ, ціле число в межах від 1 до 30.

2.1.2 Розбиття Випадкового Графа

Параметри генерації графа

Опис вхідних даних:

- **vertices** — кількість вершин, число в межах від 4 до 20;
- **edges** — кількість ребер, число в межах від кількість вершин мінус один до кількість вершин помножена на три.

Параметри бджолиного алгоритму

Опис вхідних даних:

- **ns** — кількість бджіл-розвідників, ціле число в межах від 4 до 40;

- **mb** — кількість зон концентраціїї розв'язків, ціле число в межах від 2 до кількісті бджіл-розвідників;
- **nf** — кількість бджіл-фуражирів, ціле число в межах від кількості зон концентрації розв’язків помноженої на два до кількості бджіл-розвідників помноженої на п’ять;
- **r** — "різність" бджіл фуражирів (кількість змін у бджолах-фуражирах), ціле число в межах від 1 до чверті кількості вершин у графі;
- **stopCount** — допустима кількість ітерацій без зміни значення ЦФ, ціле число в межах від 1 до 30.

2.1.3 Порівняльний Аналіз Алгоритмів Розбиття

Опис вхідних даних:

- **step —** крок збільшення розмірності задачі, число в межах від 1 до 5.

2.2 Вихідні дані

2.2.1 Розбиття Графа-приклада

Опис вихідних даних:

- **вага розрізу за ФМ** — сумарна вага мінімального розрізу знайденого алгоритмом Федуччі-Маттеуса;
- **вага розрізу за Бджолиним** — сумарна вага мінімального розрізу знайденого Бджолиним алгоритмом.

До вихідних даних належить також таблиця формату:

- **І вершина ребра** — у комірці зазначено першу вершину ребра;
- **ІІ вершина ребра** — у комірці зазначено другу вершину ребра;
- **вага ребра** — у комірці вказано вагу ребра між вершинами І та ІІ;
- **ФМ** — якщо ребро входить до розрізу, знайденого за алгоритмом Федуччі-Матеуса, то у комірці отримаємо "+", інакше "-";

– **Бджолиний** — якщо ребро входить до розрізу, знайденого за Бджолиним алгоритмом, то у комірці отримаємо "+", інакше "-".

2.2.2 Розбиття Випадкового Графа

Опис вихідних даних:

– **вага розрізу за ФМ** — сумарна вага мінімального розрізу знайденого алгоритмом Федуччі-Матеуса;
– **вага розрізу за Бджолиним** — сумарна вага мінімального розрізу знайденого Бджолиним алгоритмом;
– **кількість вершин** — кількість вершин у згенерованому графі;
– **кількість ребер** — кількість ребер у згенерованому графі.

До вихідних даних належить також таблиця формату:

– **І вершина ребра** — у комірці зазначено першу вершину ребра;
– **ІІ вершина ребра** — у комірці зазначено другу вершину ребра;
– **вага ребра** — у комірці вказано вагу ребра між вершинами І та ІІ;
– **«ФМ»** — якщо ребро входить до розрізу, знайденого за алгоритмом Федуччі-Маттеуса, то у комірці отримаємо "+", інакше "-";
– **«Бджолиний»** — якщо ребро входить до розрізу, знайденого за Бджолиним алгоритмом, то у комірці отримаємо "+", інакше "-".

До вихідних даних належить також дві діаграми:

– діаграма зміни ЦФ (вага розрізу) в залежності від ітерації алгоритму Федуччі-Маттеуса;
– діаграма зміни ЦФ (вага розрізу) в залежності від ітерації Бджолиного алгоритму.

2.2.3 Порівняльний Аналіз Алгоритмів Розбиття

Опис вихідних даних:

- гістограма залежності часу роботи алгоритму від розмірності задачі;
- Діаграма зміни ЦФ (вага розрізу) в залежності від ітерації алгоритму Федуччі-Маттеуса;
- Діаграма зміни ЦФ (вага розрізу) в залежності від ітерації Бджолиного алгоритму;
- Excel-файл з таблицями розв’язку (структура кожної з таблиць наведена у підрозділі 2.2.2) та значеннями цільових функцій для кожного графу з множини задач.

2.3 Опис інформаційного забезпечення

Є можливість запису результатів розв’язання множини задач до excel-файлу (структура файлу наведена у підрозділі 2.2.3).

3 РІШЕННЯ З ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис вибраних технологій розробки

Програмний продукт розроблено за допомогою шаблону проектування ASP.NET MVC 5 мовою програмування C#. З використанням сторонніх бібліотек:

- Bootstrap 4;
- jQuery - 3.0.0;
- Popper.js;
- Calabonga.Xml.Exports.

3.2 Архітектура програмного забезпечення

3.2.1 Схема архітектури ПЗ

Архітектуру програмного забезпечення побудовано відповідно до шаблону проектування MVC. За шаблоном застосування містить моделі, представлення та контролери.

Архітектура застосунку складається з шару представлення, бізнес шару та шару даних. Наведемо діаграму взаємозв’язку шарів архітектури (рисунок 3.1).

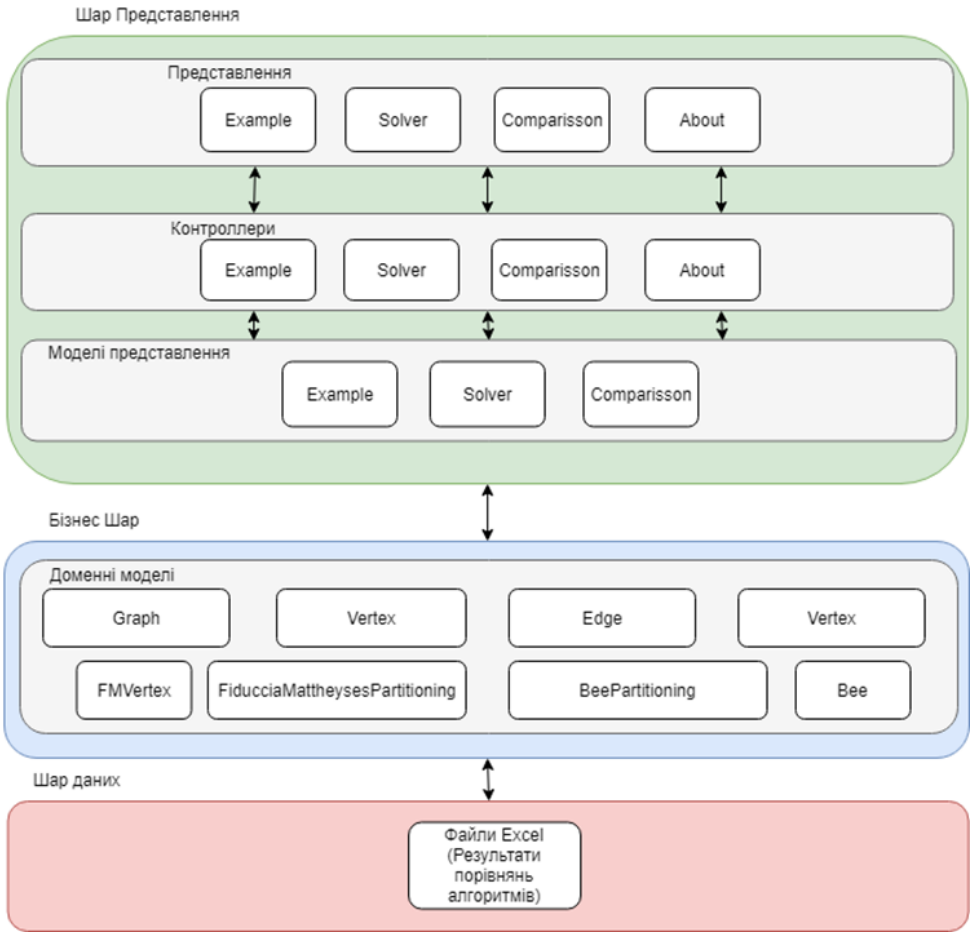

Рисунок 3.1 — Діаграма взаємозв’язку шарів архітектури

Згідно з структурою моделі MVC наведемо більш детальну інформацію про кожну групу компонент.

Контролери:

- AboutController.cs — контролер для роботи з представленням About;
- ComparisonController.cs — контролер для роботи з представленням Comparison для побудови діаграм зміни значення ЦФ в залежності від номеру ітерації та побудови гістограми залежності часу роботи алгоритмів від розмірності задачі;
- ExampleController.cs — контролер для роботи з представленням Example для побудови таблиці результату;
- SolverController.cs — контролер для роботи з представленням Solver для побудови таблиці результату та побудови діаграм зміни значення ЦФ в залежності від номеру ітерації.

Моделі представлення:

- Solver.cs — модель для представлення Solver;
- Comparison.cs — модель для представлення Comparison;
- Example.cs — модель для представлення Example.

Доменні моделі:

- Graph.cs — генерація графа;
- Edge.cs — модель ребра;
- Vertex.cs — модель вершини;
- FMVertex.cs — модель вершини з необхідними для алгоритму Федуччі-Маттеуса додатковими параметрами;
- Bee.cs — модель рішення задачі бджолиним алгоритмом;
- BeePartitioning.cs — реалізація бджолиного алгоритму;
- Fiduccia_Mattheyses.cs — реалізація алгоритму Федуччі-Маттеуса.

В свою чергу шар даних містить файли Excel файли, доступні для завантаження.

3.2.2 Діаграма класів

Наведемо діаграму класів для моделей (рисунок 3.2) з відповідними методами.

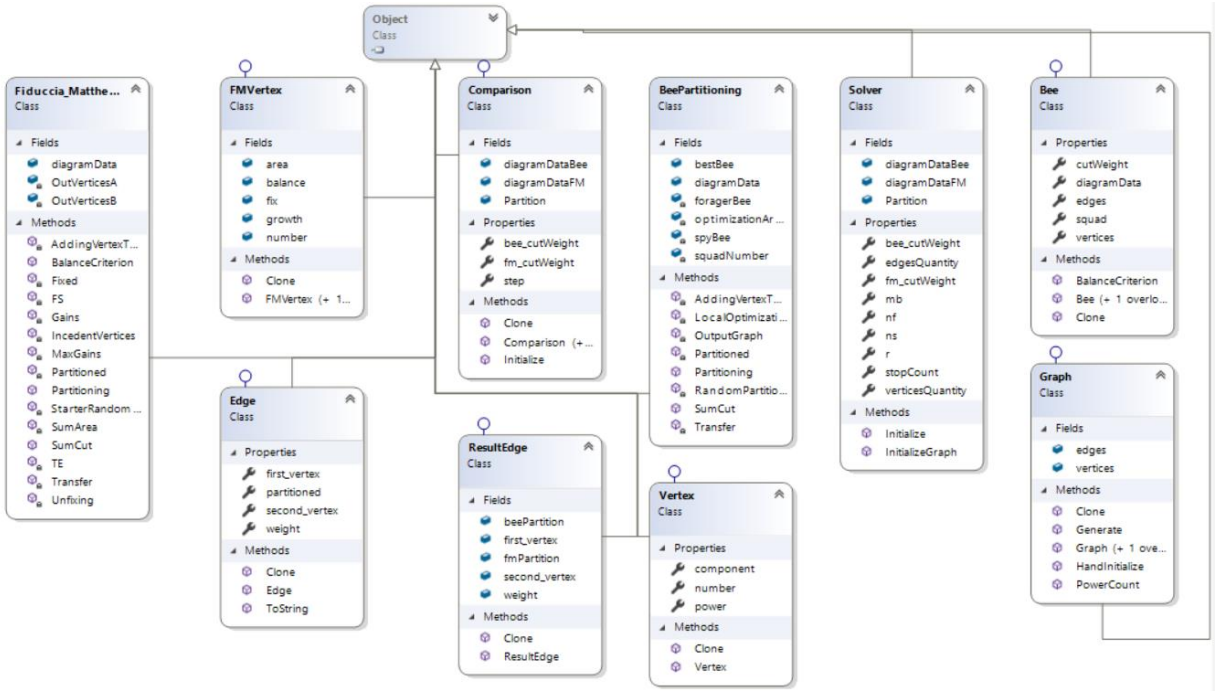

Рисунок 3.2 — Діаграма класів для моделей

Наведемо діаграму класів для контролерів (рисунок3.3) з відповідними методами.

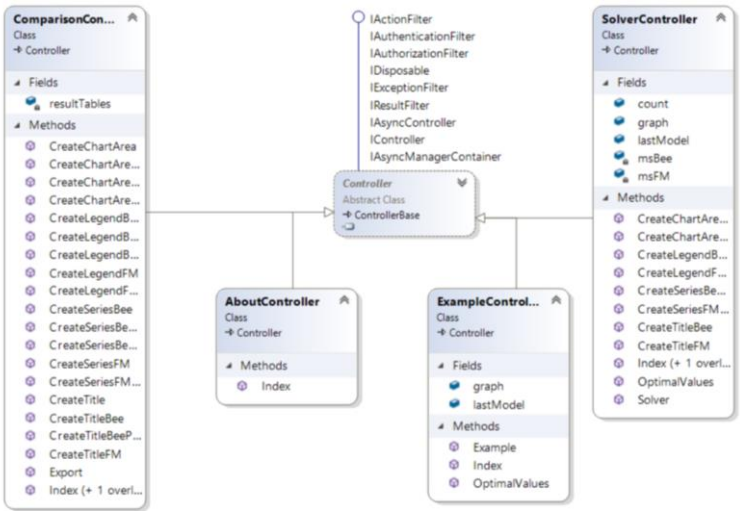

Рисунок 3.3 — Діаграма класів для контролерів

3.2.3 Діаграма послідовності

Наведемо діаграми послідовностей для основних методів FeduccіaMattheysesPartitioning та BeePartitioning.

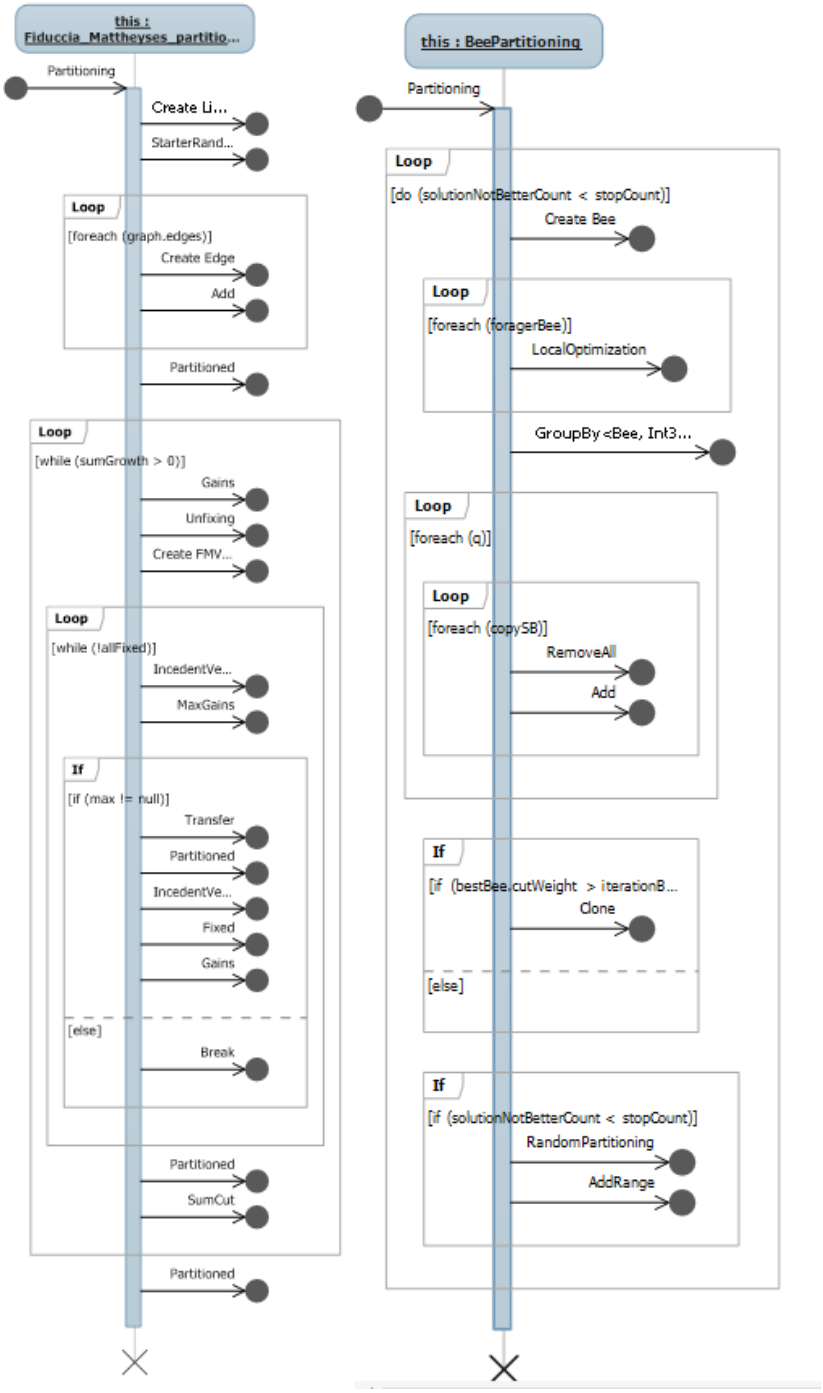

Рисунок 3.4 — Діаграми послідовностей

3.2.4 Діаграма розгортання

Наведемо діаграму розгортання застосунку (рисунок 3.5).

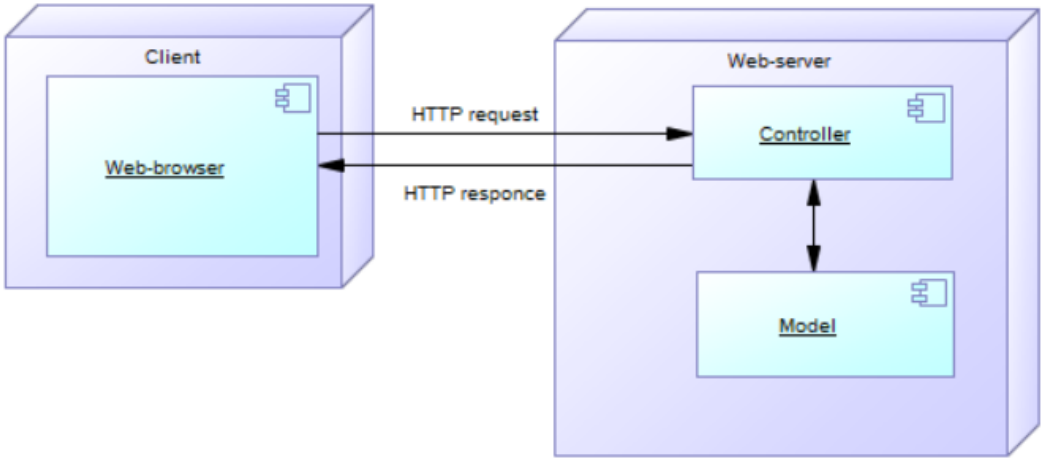

Рисунок 3.5 — Діаграма розгортання

3.2.5 Специфікація функцій

Специфікація реалізованих функцій наведена у вигляді таблиці (таблиця 3.1).

Таблиця 3.1 — Специфікація функцій

Назва функції	Опис
Введення даних	Вхідні дані вводяться за допомогою текстових полів. Дані проходять валідацію. У випадку некоректних даних виводиться повідомлення про помилку. При введенні значення у певне текстове поле з’являється відповідна йому pop-up підказка про допустимий діапазон значень.
Генерація випадкової задачі.	При правильно заповнених полях вхідних даних, по натиску на кнопку «Згенерувати» генерується нова задача (випадковий граф у вигляді таблиці списку ребер).

Продовження таблиці 3.1

Виконання алгоритмів	При правильно заповнених полях вхідних даних, по натиску на кнопку «Почати» на сторінці «Розбиття випадкового графа», виконується розв’язок відповідної задачі двома алгоритмами. Результати представляються у вигляді таблиці та значень ваг розрізів, отриманих кожним з алгоритмів.
Побудова графіків зміни результату у ході роботи алгоритмів.	Коли задача розв’язана, по натиску на кнопку «Графіки» на сторінці «Розбиття випадкового графа», зображуються графіки зміни значень розрізу від ітерації до ітерації в процесі розв’язку.
Побудова множини випадкових задач.	При правильно заповнених полях вхідних даних, по натиску на кнопку «Почати» на сторінці «Порівняльний аналіз алгоритмів розбиття», виконується генерація та розв’язок множини випадкових задач різної розмірності.
Побудова графіків порівняння швидкодії алгоритмів	Після проведення експерименту «Порівняльний аналіз алгоритмів розбиття» сторінка містить графік швидкодії у вигляді гістограми залежності часу роботи алгоритмів від розмірності задачі.
Завантаження Excel документу з розв’язками множини задач.	Після проведення експерименту «Порівняльний аналіз алгоритмів розбиття», по натиску на кнопку «Завантажити» на пристрій завантажуються розв'язки задач, які розглядаються на гістограмі (у форматі .xls).

3.2.6 Вимоги до якості

Наведемо вимоги до якості програмного забезпечення (нефункціональні вимоги):

– сторінки веб-застосунку повинні адаптуватися під роздільну здатність дисплею;

– на кожній сторінці веб-застосунку повинно бути доступним меню переходів до інших сторінок веб-застосунку;

– сторінки веб-застосунку повинні бути оформлені українською мовою;

– сторінки веб-застосунку повинні використовувати системні шрифти;

– необхідним обладнанням для роботи системи є комп’ютер з встановленим веб-браузером та сервер на якому відбувається обробка введених параметрів. Система повинна використовувати протокол для обміну даними з сервером HTTP;

– для роботи системи на комп’ютерах повинна бути встановлена операційна система Windows XP або новіша версія, операційна системи на основі Linux 3.1 чи новіша версія, MacOs 10.4 чи новіша версія зі встановленим браузером з підтримкою HTML, CSS, JavaScript (Google Chrome, Mozilla Firefox, Opera).

4 КЕРІВНИЦТВО КОРИСТУВАЧА

4.1 Інструкція користувача

У браузері відкривається нове вікно зі сторінкою «Про програму». Для переходу на інші сторінки можна обрати один з варіантів панелі навігації. Склад панелі навігації: «Розбиття випадкового графа», «Порівняльний аналіз алгоритмів розбиття», «Розбиття графа-прикладу», «Про програму».

Про програму

Сторінка, що містить інформацію про тему вирішуваної задачі та розробників даної програми (рисунок 4.1.).

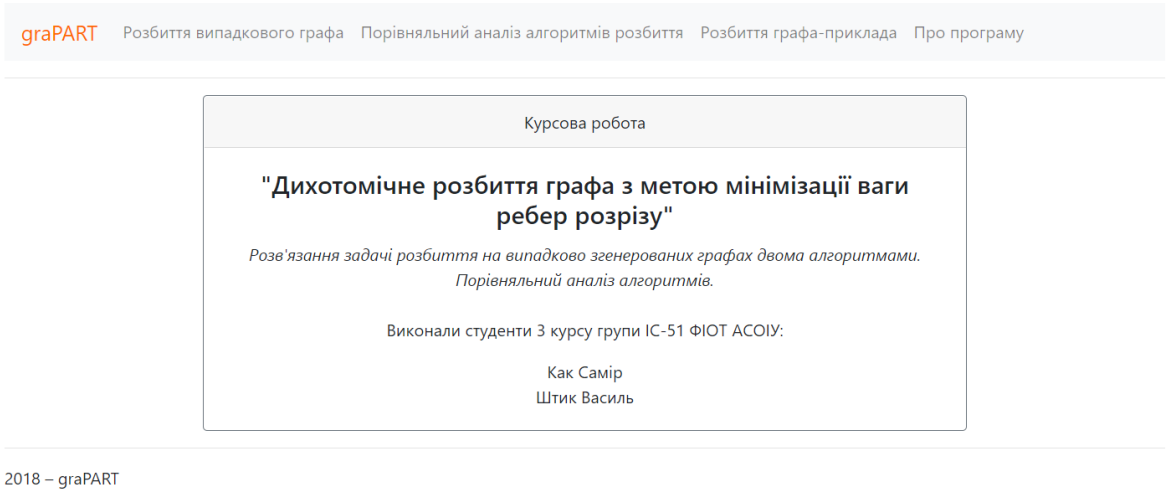

Рисунок 4.1 — Сторінка «Про програму»

Розбиття випадкового графа

Сторінка, що містить поля для введення вхідних параметрів генерації графу та довідкові відомості (рисунок 4.2). Для генерації задачі необхідно натиснути кнопку «Згенерувати». Після генерації задачі таблиця результатів заповнюється ребрами згенерованого графу, стають доступні поля введення параметрів бджолиного алгоритму та з’являється кнопка «Заповнити» для заповнення параметрів бджолиного алгоритму універсальними значеннями, довідкові відомості стають доступні за натисканням кнопки «Довідка»

(рисунок 4.3). Для запуску пошуку рішення задачі необхідно натиснути кнопку «Почати». Після знаходження рішення, таблиця результатів заповнюється даними рішення та з’являється кнопка «Графіки», при натисканні на яку розгортається область з графіками зміни значень ЦФ в залежності від номеру ітерації для обох алгоритмів (рисунок 4.4).

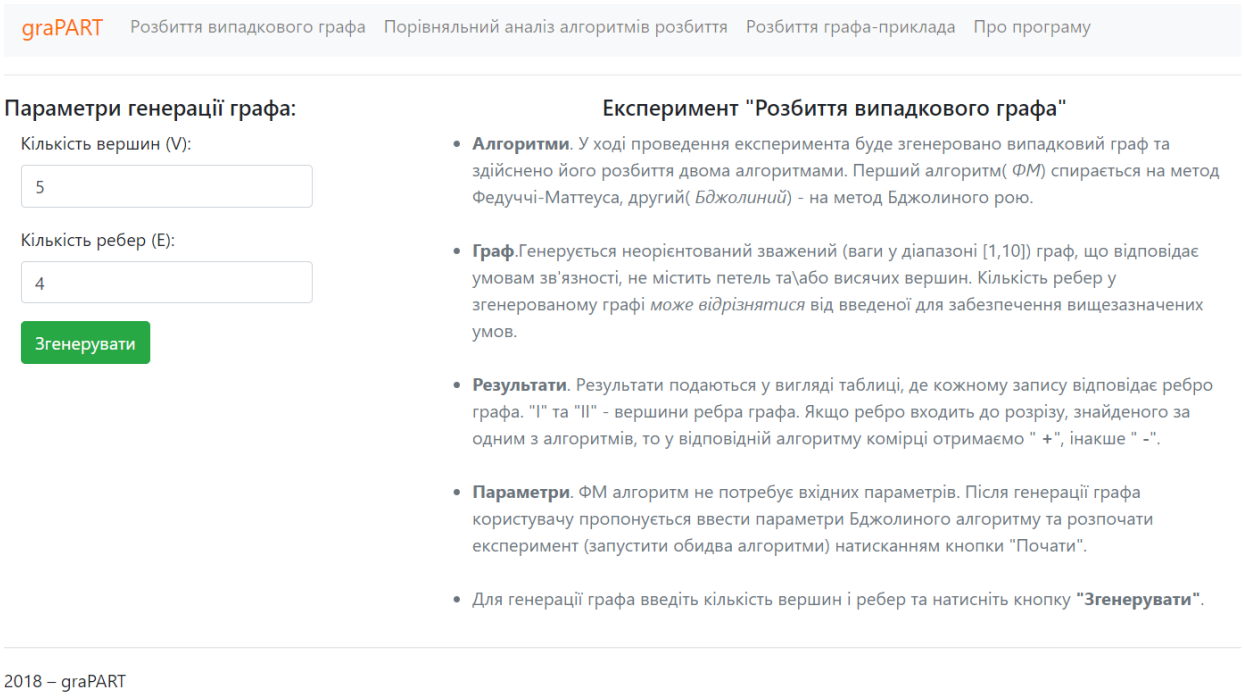

Рисунок 4.2 — Сторінка «Розбиття випадкового графа»

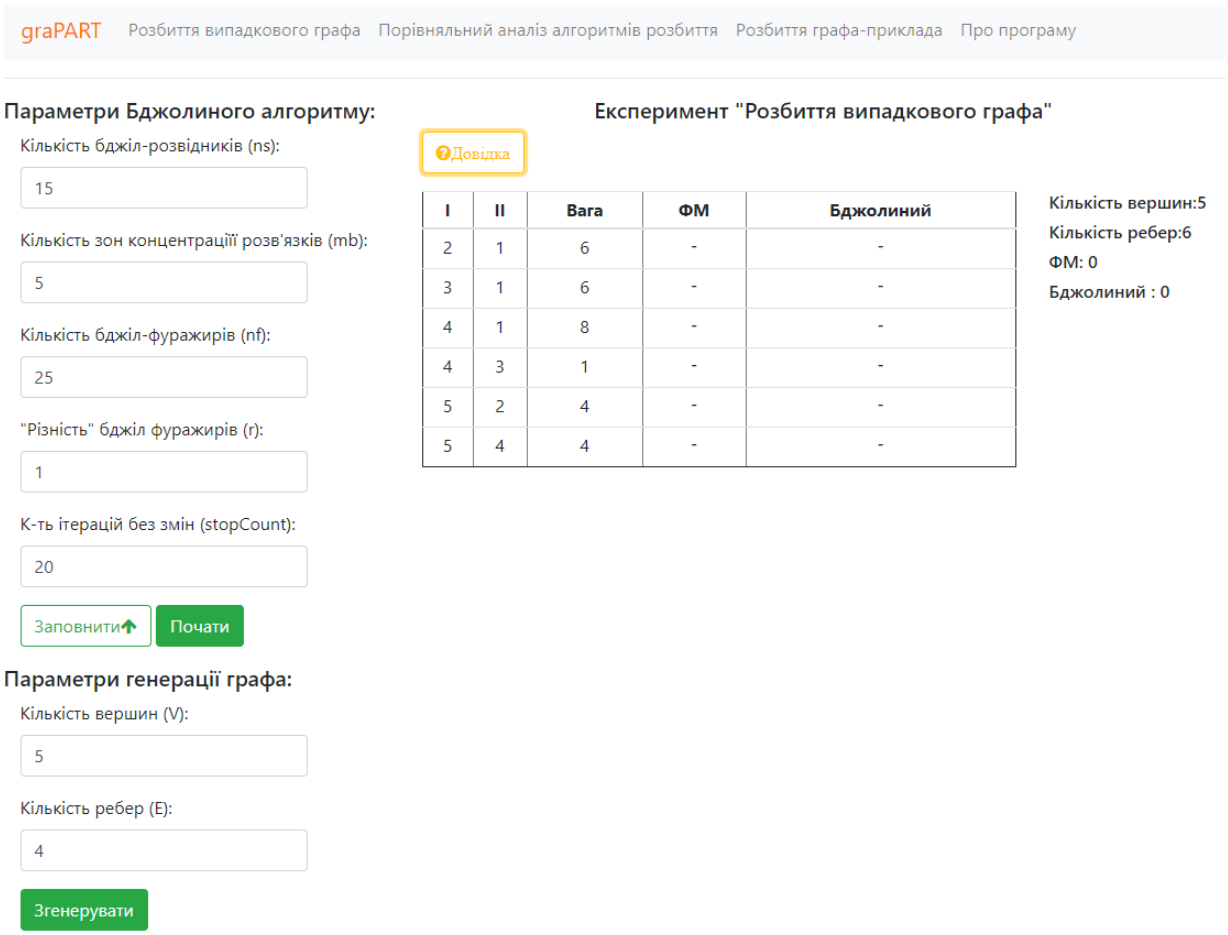

Рисунок 4.3 — Сторінка «Розбиття випадкового графа» після натискання кнопок «Згенерувати» та «Заповнити»

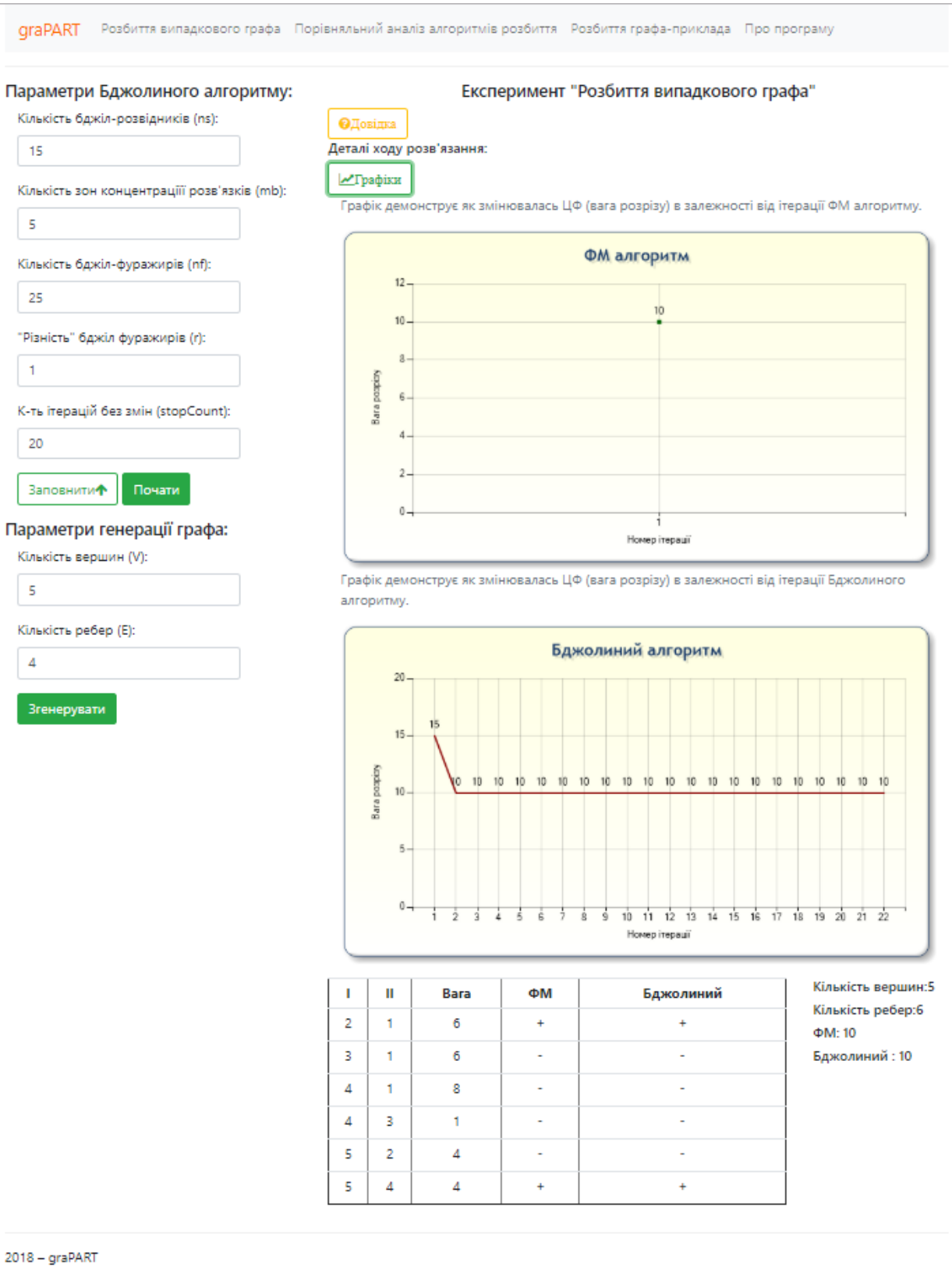

Рисунок 4.4 — Сторінка «Розбиття випадкового графа» після натискання кнопки «Почати» та з розгорнутою областю «Графіки»

Порівняльний аналіз алгоритмів розбиття

Сторінка, що містить поле для введення кроку збільшення розмірності задач та довідкові відомості (рисунок 4.5). Для генерації множини задач та їх вирішення необхідно натиснути кнопку «Почати». Після генерації та вирішення множини задач будується гістограма залежності часу роботи алгоритмів від розмірності задач(рисунок 4.6). При натисканні на кнопку «Графіки» розгортається область з графіками зміни значень ЦФ задачі найбільшої розмірності в залежності від номеру ітерації для обох алгоритмів. При натисканні на кнопку «Завантажити» почнеться скачування Excel файлу з таблицями результатів розбиття кожної задачі з множини.

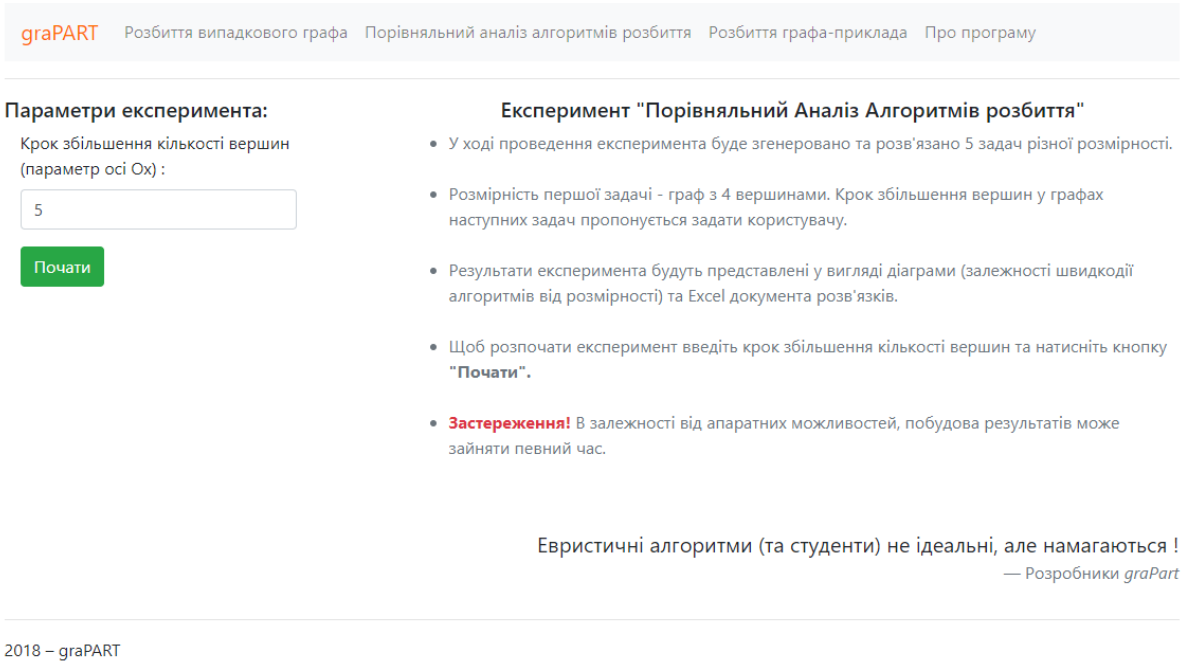

Рисунок 4.5 — Сторінка «Порівняльний аналіз алгоритмів розбиття»

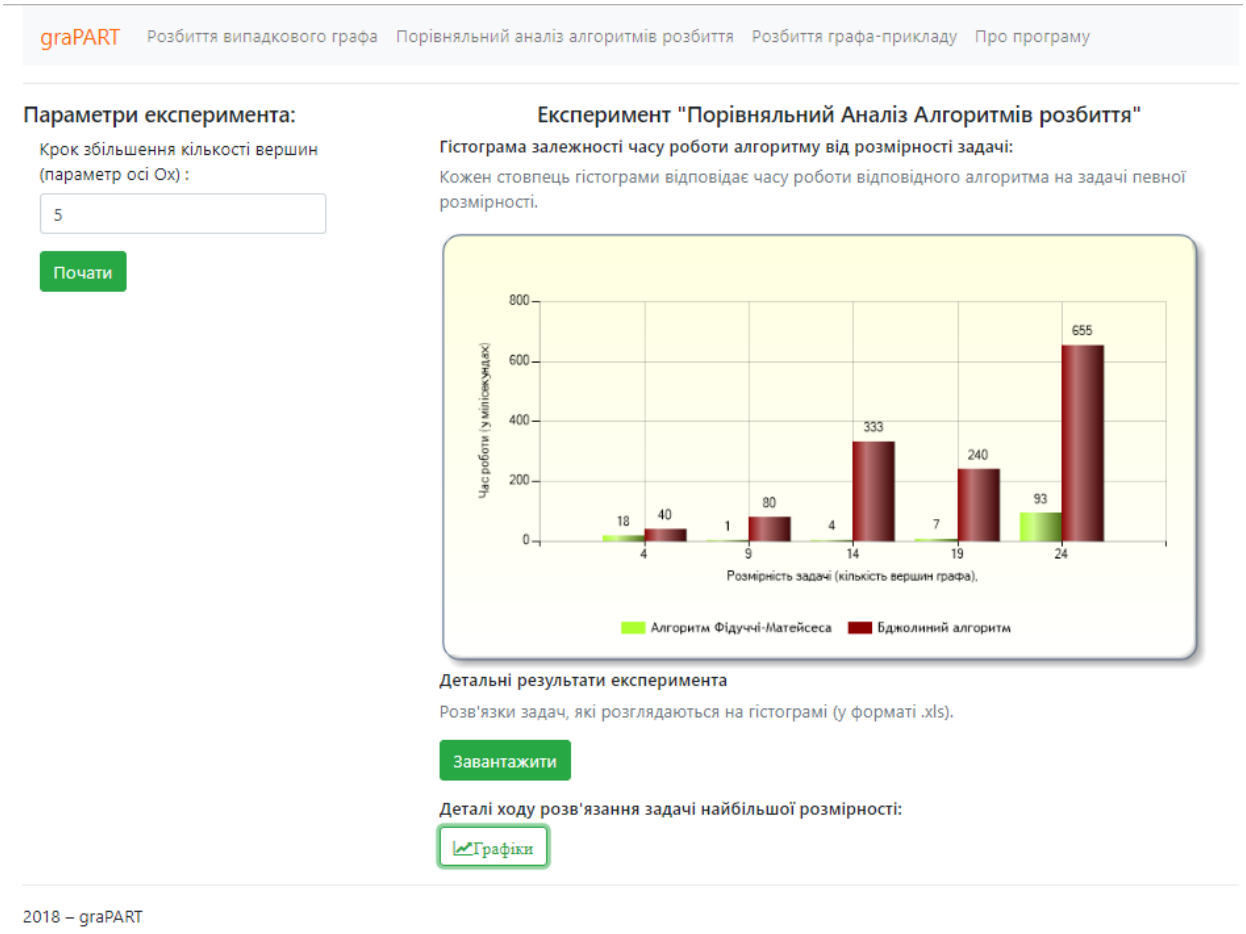

Рисунок 4.6 — Сторінка «Порівняльний аналіз алгоритмів розбиття» після натискання кнопки «Почати»

Розбиття графа-прикладу

Сторінка, що містить зображення графа-прикладу, таблицю результатів заповнену ребрами графа-прикладу та поля введення параметрів бджолиного алгоритму (рисунок 4.7). При натисканні кнопки «Заповнити» поля заповняться універсальними значеннями. Для запуску пошуку рішення задачі необхідно натиснути кнопку «Почати». Після знаходження рішення таблиця результатів заповнюється даними рішення (рисунок 4.8.).

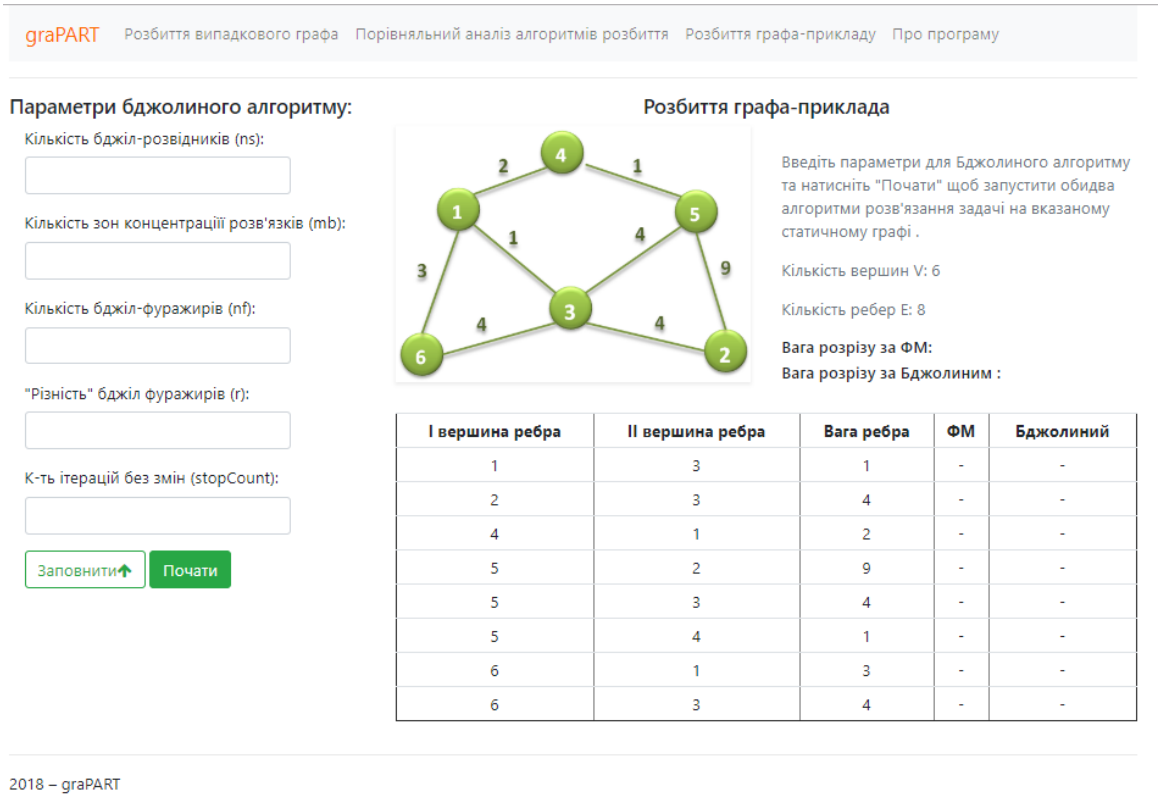

Рисунок 4.7 — Сторінка «Розбиття графа-приклада»

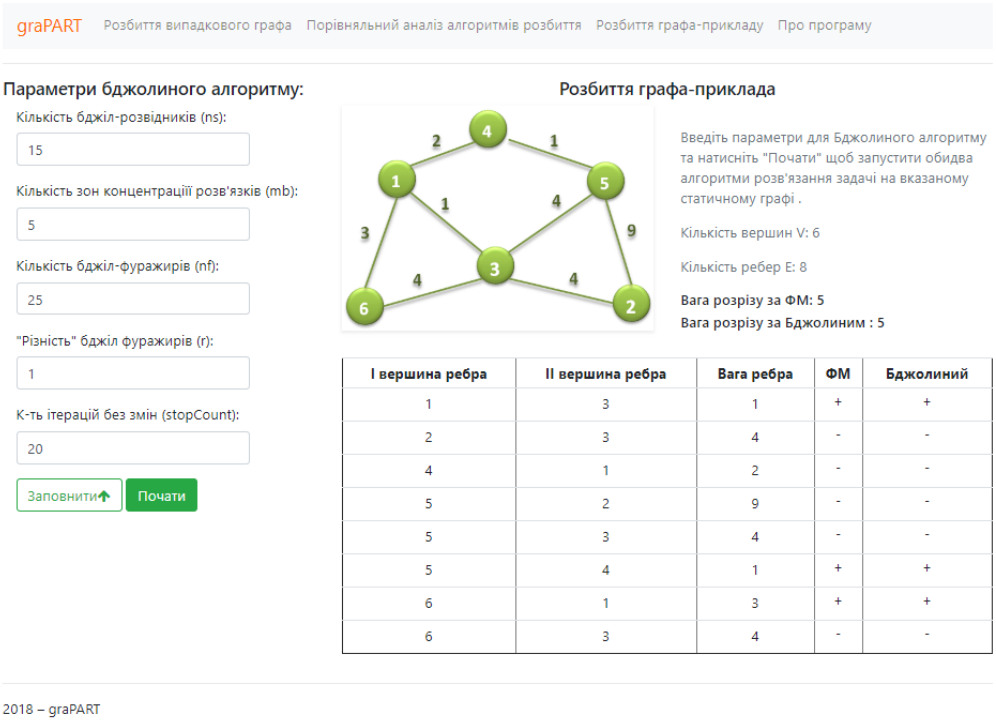

Рисунок 4.8 — Сторінка «Розбиття графа-приклада» після введення даних та натискання кнопки «Почати»

4.2 Методика випробувань

Представимо результати функціонального тестування у вигляді таблиць 4.1 — 4.7.

Таблиця 4.1 — Тестування функції «Введення даних»

Дія	Результат
Всі поля пусті, натиск на кнопку «Згенерувати» \ «Почати»	Повідомлення про пусте поле
Принаймні одне поле пусте, натиск на кнопку «Згенерувати» \ «Почати»	Повідомлення про пусте поле
Принаймні одне поле заповнене нецілим числом, натиск на кнопку «Згенерувати» \ «Почати»	Повідомлення про некоректність вхідних даних
Принаймні одне поле заповнене від'ємним числом, натиск на кнопку «Згенерувати» \ «Почати»	Повідомлення про некоректність вхідних даних
Принаймні одне поле заповнене числом, що не входить до допустимого діапазону. *Примітка: діапазони допустимих значень вказані у pop-up підказках, які з'являються при введенні значення у поле.	Повідомлення про некоректність вхідних даних
Вхідні дані коректні, натиск на кнопку «Згенерувати» \ «Почати»	Виведення результатів генерації \ розв'язку.

Таблиця 4.2 — Тестування функції «Генерація випадкової задачі»

Дія	Результат
Натиск на кнопку «Розбиття випадкового графа» головного меню	Завантаження сторінки експерименту «Розбиття випадкового графа»
Введені вхідні дані коректні, натиск на кнопку «Згенерувати»	Завантаження сторінки з згенерованою задачею.
Введені вхідні дані коректні, повторний натиск на кнопку «Згенерувати»	Завантаження сторінки з новою згенерованою задачею.

Таблиця 4.3 — Тестування функції «Виконання алгоритмів»

Дія	Результат
Задачу згенеровано, введені вхідні дані для Бджолиного алгоритму коректні, натиск на кнопку «Почати»	Виведення розв’язку задачі у вигляді таблиці та значень ваг розрізів, отриманих кожним з алгоритмів.
Задачу згенеровано, введені інші коректні вхідні дані для Бджолиного алгоритму, натиск на кнопку «Почати»	Виведення нового розв’язку задачі у вигляді таблиці та значень ваг розрізів, отриманих кожним з алгоритмів.
Кнопкою «Згенерувати», згенеровано іншу задачу, натиск на кнопку «Почати»	Виведення розв’язку нової задачі у вигляді таблиці та значень ваг розрізів, отриманих кожним з алгоритмів.

Таблиця 4.4 — Тестування функції «Побудова графіків зміни результату у ході роботи алгоритмів»

Дія	Результат
Експеримент «Розбиття випадкового графа» здійснено, натиск на кнопку «Графіки»	Розгортання області з графіками зміни значень розрізу від ітерації до ітерації в процесі розв’язку кожним з алгоритмів

Таблиця 4.5 — Тестування функції «Побудова множини випадкових задач»

Дія	Результат
Натиск на кнопку «Порівняльний аналіз алгоритмів розбиття» головного меню	Завантаження сторінки експерименту «Порівняльний аналіз алгоритмів розбиття»
Введені вхідні дані коректні, натиск на кнопку «Почати»	Завантаження сторінки аналізу розв’язку згенерованої множини задач

Таблиця 4.6. Тестування функції «Завантаження Excel документу з розв’язками множини задач»

Дія	Результат
Експеримент «Порівняльний аналіз алгоритмів розбиття» проведено, натиск на кнопку «Завантажити»	Завантаження розв’язку множини задач на пристрій користувача у вигляді Excel файлу.

Таблиця 4.7 — Тестування функції «Побудова графіків порівняння швидкодії алгоритмів»

Дія	**Результат**
Перший \ Повторний натиск на кнопку «Почати» сторінки експерименту «Порівняльний аналіз алгоритмів розбиття»	Завантаження сторінки, що містить гістограми залежності часу роботи алгоритмів від розмірності задачі.

ВИСНОВОК

В результаті виконання курсової роботи розроблено веб-застосунок «graPART», що може бути використаним для розв’язання задачі або множини задач дихотомічного розбиття графу з метою мінімізації ваги розрізу, генерації такої задачі або множини задач, побудови гістограми залежності часу розв’язання задачі від її розмірності та побудови графіку покращення значення ЦФ на кожній ітерації. Для вирішення задачі програмний продукт використовує алгоритм Федуччі-Маттеуса та бджолиний алгоритм [4].

Архітектура веб-застосунку побудована відповідно до шаблону проектування MVC. Система розділена на три взаємопов'язані частини: моделі, представлення та контролери, при цьому зміни у представленнях мінімально впливають на моделі, а зміни в моделях можуть здійснюватися без змін представлень. Веб застосунок реалізовано за допомогою фреймворку ASP.NET MVC 5 мовою програмування C#, з використанням сторонніх бібліотек: Bootstrap 4, jQuery - 3.0.0, Popper.js та Calabonga.Xml.Exports.

Під час роботи над застосунком ми провели аналіз роботи алгоритмів, оцінили їх трудомісткость, визначили часову складність, порівняли час розв’язання задач різної розмірності на основі побудованих гістограм та порівняли ефективності алгоритмів на основі графіків зміни значення ЦФ в залежності від номеру ітерації.

У пояснювальній записці було описано архітектуру програмного забезпечення, наведено діаграму взаємодії шарів архітектури, діаграми класів для моделей та контролерів, діаграми послідовностей та діаграму розгортання. Описано інструкцію користувача з використання веб-застосунку та зроблено висновки щодо роботи.

Надано перелік посилань, використаних при дослідженні задачі дихотомічного розбиття графа, розробці та реалізації алгоритмів вирішення задачі.

Програмний код основних класів наведено у додатку Б.

ПЕРЕЛІК ПОСИЛАНЬ

1. VLSI Physical Design: From Graph Partitioning to Timing Closure / Andrew B. Kahng, Jens Lienig, Igor L. Markov, Jin Hu // Springer, 2011. – 309 с.

2. Биоинспирированные методы в оптимизации / Л. Гладков, В.Курейчик, В.Курейчик, П. Сороколетов // М.:ФИЗМАТЛИТ, 2009 – 384 с.

3. Graph Partitioning and Graph Clustering in Theory and Practice / C. Schulz: Course Notes // Institute for Theoretical Informatics Karlsruhe Institute of Technology (KIT), 2016. – 201 с.

4. Естественные алгоритмы. Алгоритм поведения роя пчёл // [Електронний ресурс] — Режим доступу: https://habr.com/post/104055/

ДОДАТОК А

А.1. Календарний план

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ ПРОЕКТУ

№	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Примітка
1	*Опис постановки задачі*	*01.03*	
2	*Консультація у керівника*	*05.03*	
3	*Пошук літературних джерел за темою курсової роботи. Порівняльний аналіз існуючих методів розв'язання задачі*	*15.03*	
4	*Вивчення теоретичних положень метода Федуччі-Маттеуса та метаевристичних підходів до розв'язання задачі*	*31.03*	
5	*Розробка алгоритму Федуччі-Маттеуса розв'язання задачі.*	*05.04*	
6	*Розробка метаевристичного алгоритму розв'язання задачі*	*15.04*	
7	*Консультація в керівника – погодження інтерфейсу програми*	*25.04*	
8	*Розробка програмної реалізації. Відлагодження програм. Визначення найбільш ефективних стратегій методу Федуччі-Маттеуса та метаевристичного*	*10.05*	
9	*Оцінка складності алгоритмів розв'язання задачі*	*12.05*	
10	*Оформлення пояснювальної записки*	*14.05*	
11	*Захист роботи*	*17.05*	

А.2. Початковий план робіт

Пункт	**Початок**	**Дні виконання**	**Кінцевий термін**	**Виконавець**
Побудова математичної моделі	05.03.2018	5	10.03.2018	Как С.Р., Штик В.Л.
Аналіз можливих методів розв'язання	10.03.2018	10	20.03.2018	Как С.Р., Штик В.Л.
Розробка алгоритму Федуччі-Маттеуса	20.03.2018	7	27.03.2018	Как С.Р.,
Розробка бджолиного алгоритму	27.03.2018	7	02.04.2018	Как С.Р.,
Реалізація алгоритму Федуччі-Маттеуса	02.03.2018	10	12.04.2018	Штик В.Л
Реалізація бджолиного алгоритму	12.04.2018	10	22.04.2018	Штик В.Л
Реалізація генератору задач	22.04.2018	5	27.04.2018	Штик В.Л.
Реалізація веб-застосунку	27.04.2018	11	08.05.2018	Как С.Р.
Створення ПЗ	08.05.2018	9	16.05.2018	Как С.Р., Штик В.Л.
Захист проекту	17.05.2018	1	17.05.2018	Как С.Р., Штик В.Л.

А.3. Кінцевий план робіт

Пункт	**Початок**	**Дні виконання**	**Кінцевий термін**	**Виконавець**
Побудова математичної моделі	05.03.2018	5	10.03.2018	Как С.Р., Штик В.Л.
Аналіз можливих методів розв'язання	10.03.2018	10	20.03.2018	Как С.Р., Штик В.Л.
Розробка алгоритму Федуччі-Маттеуса	20.03.2018	7	27.03.2018	Штик В.Л.
Розробка бджолиного алгоритму	27.03.2018	8	03.04.2018	Как С.Р.
Реалізація алгоритму Федуччі-Маттеуса	03.03.2018	5	08.04.2018	Штик В.Л.
Реалізація бджолиного алгоритму	08.04.2018	10	18.04.2018	Как С.Р.
Реалізація генератору задач	18.04.2018	6	24.04.2018	Штик В.Л.
Реалізація веб-застосунку	24.04.2018	14	08.05.2018	Как С.Р., Штик В.Л.
Створення ПЗ	08.05.2018	9	16.05.2018	Как С.Р., Штик В.Л.
Захист проекту	17.05.2018	1	17.05.2018	Как С.Р., Штик В.Л.

ДОДАТОК Б

Б.1 Тексти програмного коду

BeePartitioning.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Grapart_MVC
{
    public class BeePartitioning//Бджолиний алгоритм
    {
        List<Bee> spyBee = new List<Bee>();
        List<Bee> optimizationAreas = new List<Bee>();
        List<Bee> foragerBee = new List<Bee>();
        public List<int> diagramData = new List<int>();
        public Bee bestBee = new Bee();
        int squadNumber = 0;

        public Bee Partitioning(Graph graph, int ns, int mb, int nf, int r, int
stopCount)//Розбиття за Бджолиним алгоритмом
        {
            spyBee.Clear();
            optimizationAreas.Clear();
            foragerBee.Clear();
            diagramData.Clear();
            bestBee.diagramData= new List<int>();
            squadNumber = 0;
            int solutionNotBetterCount = 0;

            //генерируем начальное множество пчел-разведчиков
            spyBee.AddRange(RandomPartitioning(graph, ns));

            bestBee = spyBee.Clone().First();
            Console.WriteLine("FirstBee"+'\n');
            OutputGraph(bestBee, 1);

            do
            {
                Bee iterationBestBee = new Bee();

                //упорядывачием пчел-разведчиков по значению cutWeight(сумарный вес
разреза)
                //в участки оптимизации добавляем mb лучших пчел-разведчиков
                optimizationAreas = new List<Bee>(spyBee.Clone().OrderBy(e =>
e.cutWeight).Take(mb));
                spyBee = new List<Bee>(optimizationAreas.Clone());

                int y = nf;

                //распредиление пчел-фуражиров по участкам оптимизации
                #region Peresmotret'
                //пока не кончились пчелы-фуражиры
```

```
                while (y > 0)
                {
                    //добовляем по одной пчеле на каждый участок
                    foreach (var item in optimizationAreas)
                    {
                        if (item == optimizationAreas.ElementAt(0))
                        {
                            foragerBee.Add((Bee)item.Clone());
                        }
                        y--;
                        foragerBee.Add((Bee)item.Clone());
                        if (y <= 0)
                        {
                            break;
                        }
                    }
                }
                #endregion

                //проводим локальное изменение пчел-фуражиров
                foreach (var item in foragerBee)
                {
                    LocalOptimization(item, r);
                }

                //групируем пчел-фуражиров по отрядам(squad)
                var q = from enf in foragerBee
                        group enf by enf.squad;

                List<Bee> copySB = new List<Bee>(spyBee.Clone());

                foreach (var item in q)
                {
                    //в текущем отряде отбираем лучшую пчелу-фуражира
                    var cell = item.OrderBy(e => e.cutWeight).First();
                    //ищем пчелу-разведчика соответсвующую текущему отряду
                    foreach (var element in copySB)
                    {
                        if (element.squad == cell.squad)
                        {
                            //если пчела-фуражир лучше своей пчелы-разведчика
                            if (element.cutWeight > cell.cutWeight)
                            {
                                //пчела-фуражир заменяет пчелу-разведчика
                                spyBee.RemoveAll(e => e.squad == element.squad);
                                spyBee.Add(cell);
                            }
                            break;
                        }
                    }
                }

                //упорядывачиние решений текущей итерации и выбор лучшего
                iterationBestBee = spyBee.Clone().OrderBy(e => e.cutWeight).First();

                Console.WriteLine("iterationBestBee"+'\n');
                OutputGraph(iterationBestBee, 2);

                //если cutWeight лучшего решения текучей итерации меньше от cutWeight
общего лучшего решения
                if (bestBee.cutWeight > iterationBestBee.cutWeight)
```

```
                {
                    //лучшее решение текучей итерации становится общим лучшим
решением
                    Console.WriteLine("CommonBestBee"+'\n');
                    bestBee = (Bee)iterationBestBee.Clone();
                    //и обнуляется счетчик итераций без улучшения
                    solutionNotBetterCount = 0;
                }
                else//иначе увеличиваем счетчик итераций без улучшения
                {
                    solutionNotBetterCount++;
                }

                //если счетчик итераций без улучшения не достиг "значения остановки"
                if (solutionNotBetterCount < stopCount)
                {
                    //генерируем ns - mb новых пчел разведчиков
                    spyBee.AddRange(RandomPartitioning(graph, ns - mb));
                }
                diagramData.Add(bestBee.cutWeight);
            } while (solutionNotBetterCount < stopCount);
            bestBee.diagramData=diagramData;
            return bestBee;
        }

        private List<Bee> RandomPartitioning(Graph graph, int count)//рандомно-
направленное создание пчел(разбиений)
        {
            Bee bee = new Bee();
            List<Bee> beeList = new List<Bee>();
            int num;
            Random rand = new Random();

            for (int i = 0; i < count; i++)//создаем count пчёл
            {
                bee.edges = new List<Edge>(graph.edges.Clone());

                bee.vertices = new List<Vertex>(graph.vertices.Clone());

                num = rand.Next(1, bee.vertices.Count);

                List<Vertex> numbers= new List<Vertex>(AddingVertexToComponent(bee,
num));
                List<Vertex> numbersNew = new List<Vertex>();
                while (!bee.BalanceCriterion())
                {
                    foreach (var vertex in numbers)
                    {
                        if (bee.BalanceCriterion())
                        {
                            break;
                        }
                        numbersNew.AddRange(AddingVertexToComponent(bee,
vertex.number));
                        numbersNew.RemoveAll(e=>e.number==vertex.number);

                    }
                    numbers=(List<Vertex>)numbersNew.Clone();
                }
                Partitioned(bee);
                bee.cutWeight = SumCut(bee.edges);
```

```
                bee.squad = squadNumber++;
                beeList.Add((Bee)bee.Clone());
            }
            return beeList;
        }

        private List<Vertex> AddingVertexToComponent(Bee bee, int vertexNumber)//для
каждой вершины в компоненте добавляем ее соседей пока не виполниться КБ
        {
           List<Vertex> Neighbors=new List<Vertex>();
            bee.edges.Shuffle();
            foreach (var element in bee.edges)
            {
                if (bee.BalanceCriterion())
                {
                    break;
                }
                else if (element.first_vertex.number == vertexNumber)
                {
                    //if (element.first_vertex.component == 0)
                    //{
                        bee.vertices[vertexNumber - 1].component = 1;
                        element.first_vertex.component = 1;
                    //}
                    if (element.second_vertex.component == 0)
                    {
                        bee.vertices[element.second_vertex.number - 1].component = 1;
                        element.second_vertex.component = 1;
                        Neighbors.Add(element.second_vertex);
                    }
                }
                else if (element.second_vertex.number == vertexNumber)
                {
                    //if (element.second_vertex.component == 0)
                    //{
                        element.second_vertex.component = 1;
                        bee.vertices[vertexNumber - 1].component = 1;
                    //}
                    if (element.first_vertex.component == 0)
                    {
                        element.first_vertex.component = 1;
                        Neighbors.Add(element.first_vertex) ;
                        bee.vertices[element.first_vertex.number - 1].component = 1;
                    }
                }
            }
            return Neighbors;
        }

        public int SumCut(List<Edge> edgesList)//подсчет веса сумарного разреза
        {
            int count = 0;
            //проходимось по усім ребрам
            foreach (var element_edge in edgesList)
            {
                //у разрезаных ребер
                if (element_edge.partitioned == true)
                {
                    //прибавляем вес этого ребра к сумарному весу разреза
                    count += element_edge.weight;
                }
```

```
            }
            return count;
        }

        private void LocalOptimization(Bee bee, int r)//рандомно-направленное
изменение пчел-фуражиров
        {
            //отбор вершин инцидентных ребрам разреза
            //забираем вершины что стоят на первой позиции в ребре
            var q1 = from e in bee.edges
                     where e.partitioned == true
                     select e.first_vertex;
            //забираем вершины что стоят на второй позиции в ребре
            var q2 = from e in bee.edges
                     where e.partitioned == true
                     select e.second_vertex;
            List<Vertex> q3 = new List<Vertex>();
            q3.AddRange(q1);
            q3.AddRange(q2);


            //формируем список вершин инцидентных ребрам разреза без повторений
            var q4 = (from vertex in q3
                      select vertex.number).Distinct().ToList();

            List<Vertex> q5 = new List<Vertex>();

            foreach (var vertex in bee.vertices)
            {
                foreach (var num in q4)
                {
                    if (vertex.number==num)
                    {
                        q5.Add((Vertex)vertex.Clone());
                    }
                }

            }

            Random rand = new Random();
            bool cb = false;
            //для гарантии выполнения критерия баланса
            //прооцес изменения начинается заново пока не выполнится критерий баланса
            while (!cb)
            {
                Bee copyBee = (Bee)bee.Clone();

                //перетягиваем r рандомных вершин в противоположную компоненту
                for (int i = 0; i < r; i++)
                {
                    //если q5 не пустая
                    if (q5.Count > 0)
                    {
                        //перетягиваем рандомную вершину в противоположную компоненту

                        Vertex randVertex = q5.ElementAt(rand.Next(0, q5.Count));
                        Transfer(copyBee, randVertex);
                    }
                    else break;

                }
```

```
                //проверяем КБ для новоизмененной копии пчелы-фуражира
                cb = copyBee.BalanceCriterion();
                if (cb)
                {
                    //в случае выполнения пчела-фуражир принимает состояние копии
                    bee = (Bee)copyBee.Clone();
                    Partitioned(bee);
                    bee.cutWeight = SumCut(bee.edges);
                }

            }
        }

        private void Partitioned(Bee bee)//актуализация значений partitioned и
component у ребер
        {
            //актуализация значений component
            foreach (var vertex in bee.vertices)
            {
                foreach (var edge in bee.edges)
                {
                    if (vertex.number == edge.first_vertex.number)
                    {
                        edge.first_vertex.component=vertex.component;
                    }
                    else if (vertex.number == edge.second_vertex.number)
                    {
                        edge.second_vertex.component=vertex.component ;
                    }
                }
            }
            //актуализация значений partitioned
            foreach (var edge in bee.edges)
            {
                if (edge.first_vertex.component != edge.second_vertex.component)
                {
                    edge.partitioned = true;
                }
            }
        }

        private void Transfer(Bee bee, Vertex vertex)//перемещение вершины в
противоположную компоненту
        {

            foreach (var vert in bee.vertices)
            {
                if (vert.number==vertex.number)
                {
                    if (vertex.component == 0)
                    {
                        vert.component = 1;
                    }
                    else vert.component = 0;
                }
            }
        }

        private void OutputGraph(Bee bee,int type)//Консольний вивід
        {
            if (type == 1)
```

```
            {
                Partitioned(bee);
                Console.WriteLine("-I--II---Weight--Partitioned");
                foreach (var item in bee.edges)
                {
                    Console.WriteLine(item.ToString());
                }
                Console.WriteLine(bee.cutWeight);
            }
            if (type == 2)
            {
                Console.WriteLine(bee.squad+"  "+bee.cutWeight);
            }
        }

    }
}
```

FiduccіaMattheysesPartitioning.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Grapart_MVC
{

    public class Fiduccia_Mattheyses_partitioning //Алгоритм Федуччі-Маттеуса
    {
        List<FMVertex> OutVerticesA = new List<FMVertex>();//1-я компонента на
внешней итерации
        List<FMVertex> OutVerticesB = new List<FMVertex>();//2-я компонента на
внешней итерации
        public static List<int> diagramData = new List<int>();

        public List<Edge> Partitioning(Graph graph)//РОЗБИТТЯ за алгоритмом ФМ
        {
            List<Edge> Edges = new List<Edge>();
            int sumGrowth = 1;
            int endGrowth=0;
            int sum_area = graph.vertices.Count;
            List<FMVertex> gainsList = new List<FMVertex>();
            List<FMVertex> InVerticesA;//1-я компонента на внутринней итерации
            List<FMVertex> InVerticesB;//2-я компонента на внутренней итерации

            diagramData.Clear();
            OutVerticesA.Clear();
            OutVerticesB.Clear();

            StarterRandomPartitioning(graph);

            //bf = BalanceFactor();

            foreach (var item in graph.edges)
            {
                Edges.Add(new Edge(item.first_vertex, item.second_vertex,
item.weight, false));
            }
```

```
            Partitioned(Edges, OutVerticesA, OutVerticesB);
            //OutputGraph(Edges);
            //допоки відбувається покращення
            while (sumGrowth > 0)
            {

                gainsList.Clear();

                InVerticesA = new List<FMVertex>(OutVerticesA.Clone());

                InVerticesB = new List<FMVertex>(OutVerticesB.Clone());

                //обчислити прирости усіх вершин
                Gains(Edges, InVerticesA, InVerticesB);

                //скинути фіксацію вершин
                Unfixing(InVerticesA, InVerticesB);

                FMVertex baseVertex = new FMVertex(-1,1,0,false);
                bool allFixed= false;
                int step = 1;

                //допоки є незаіфксовані вершини інцедентні ребрам розрізу
                while (!allFixed)
                {
                    //обираємо вершину серед незафіксованих інцедентних ребрам
розрізу
                    var max = MaxGains(InVerticesA, InVerticesB, (sum_area),
IncedentVertices(Edges, InVerticesA, InVerticesB), step);
                    //якщо така вершина знайшлсь
                    if (max != null)
                    {
                        //перетягуємо до протилежної компоненти
                        Transfer(InVerticesA, InVerticesB, max);
                        //фіксуємо
                        max.fix = true;
                        baseVertex = max;
                        gainsList.Add(new FMVertex(max.number, max.area, max.growth,
max.fix));
                        Partitioned(Edges, InVerticesA, InVerticesB);
                        allFixed =Fixed(IncedentVertices(Edges, InVerticesA,
InVerticesB));

                        //оновлюємо прирости вершин
                        Gains(Edges, InVerticesA, InVerticesB);
                        step++;
                    }
                    //якщо ні виходимо
                    else break;
                }

                //розраховуємо послідовність приростів
                for (int i = 1; i < gainsList.Count; i++)
                {
                    gainsList[i].growth += gainsList[i - 1].growth;
                }

                //обираємо найкращу послідовність
```

```
                var max2 = gainsList.OrderByDescending(e => e.growth).Select(e =>
e).FirstOrDefault();

                if (max2 == null || max2.growth <= 0)
                {
                    //виходимо з алгоритму
                    sumGrowth = 0;
                    break;
                }
                else
                {
                    //застосовуємо послідовність перестановок
                    sumGrowth = max2.growth;
                    endGrowth += max2.growth;
                    int i = 0;
                    do
                    {
                        Transfer(OutVerticesA, OutVerticesB, gainsList[i]);
                        i++;
                    } while (gainsList[i-1].number!=max2.number);
                    //і починаємо ітерацію заново на зміненому графі
                }
                Partitioned(Edges, OutVerticesA, OutVerticesB);
                diagramData.Add(SumCut(Edges));
            }
            Partitioned(Edges, OutVerticesA, OutVerticesB);

            return Edges;
            //OutputGraph(Edges);
            //Console.WriteLine("CutWeight"+ SumCut(Edges));
            //Console.WriteLine("SumGrows: " + endGrowth);
        }

        private List<FMVertex> IncedentVertices(List<Edge> edges, List<FMVertex> A,
List<FMVertex> B)//возврат списка вершин инцедентных ребрам разреза
        {
            List<FMVertex> Incedent = new List<FMVertex>();
            //отбор вершин инцидентных ребрам разреза
            //забираем вершины что стоят на первой позиции в ребре
            var q1 = from e in edges
                     where e.partitioned == true
                     select e.first_vertex;
            //забираем вершины что стоят на второй позиции в ребре
            var q2 = from e in edges
                     where e.partitioned == true
                     select e.second_vertex;
            List<Vertex> q3 = new List<Vertex>();
            q3.AddRange(q1);
            q3.AddRange(q2);


            //формируем список вершин инцидентных ребрам разреза без повторений
            var q4 = (from vertex in q3
                      select vertex.number).Distinct().ToList();
            foreach (var vertexA in A)
            {
                foreach (var vert in q4)
                {
                    if (vert == vertexA.number)
                    {
                        Incedent.Add((FMVertex)vertexA.Clone());
```

```
                    }
                }

            }
            foreach (var vertexB in B)
            {
                foreach (var vert in q4)
                {
                    if (vert == vertexB.number)
                    {
                        Incedent.Add((FMVertex)vertexB.Clone());
                    }
                }

            }
            return Incedent;

        }

        private void StarterRandomPartitioning(Graph graph)//рандомно-направленное
создание начального разбиения
        {
            int num;
            Random rand = new Random();
            num = rand.Next(1, graph.vertices.Count);
            foreach (var vertex in graph.vertices)
            {
                OutVerticesA.Add(new FMVertex(vertex.number, 1));
            }
            List<FMVertex> numbers = new
List<FMVertex>(AddingVertexToComponent(graph, num));
            List<FMVertex> numbersNew = new List<FMVertex>();
            while (!BalanceCriterion(OutVerticesA))
            {
                foreach (var vertex in numbers)
                {
                    if (BalanceCriterion(OutVerticesA))
                    {
                        break;
                    }
                    numbersNew.AddRange(AddingVertexToComponent(graph,
vertex.number));
                    numbersNew.RemoveAll(e => e.number == vertex.number);

                }
                numbers = (List<FMVertex>)numbersNew.Clone();
            }
        }

        private List<FMVertex> AddingVertexToComponent(Graph graph, int
vertexNumber)//для каждой вершины в компоненте добавляем ее соседей пока не
виполниться КБ
        {
            List<FMVertex> Neighbors = new List<FMVertex>();
            foreach (var element in graph.edges)
            {
                if (BalanceCriterion(OutVerticesA))
                {
                    break;
                }
                else if (element.first_vertex.number == vertexNumber)
```

```
                {
                    if (OutVerticesB.Select(e=>e.number ==
element.first_vertex.number).Count()==0)
                    {
                        OutVerticesB.Add(new FMVertex(element.first_vertex.number,
1));
                        OutVerticesA.RemoveAll(e => e.number ==
element.first_vertex.number);
                    }
                    if (element.second_vertex.component == 0)
                    {
                        OutVerticesB.Add(new FMVertex(element.second_vertex.number,
1)); ;

OutVerticesA.RemoveAll(e=>e.number==element.second_vertex.number);
                        Neighbors.Add(new FMVertex(element.second_vertex.number, 1));
                    }
                }
                else if (element.second_vertex.number == vertexNumber)
                {
                    if (OutVerticesB.Select(e => e.number ==
element.second_vertex.number).Count() == 0)
                    {
                        OutVerticesB.Add(new FMVertex(element.second_vertex.number,
1));
                        OutVerticesA.RemoveAll(e => e.number ==
element.second_vertex.number);
                    }
                    if (element.second_vertex.component == 0)
                    {
                        OutVerticesB.Add(new FMVertex(element.first_vertex.number,
1));
                        OutVerticesA.RemoveAll(e => e.number ==
element.first_vertex.number);
                        Neighbors.Add(new FMVertex(element.first_vertex.number, 1));
                    }
                }
            }
            return Neighbors;
        }

        private void Partitioned(List<Edge> edgesList, List<FMVertex> listA,
List<FMVertex> listB)//определяет попало ли ребро под разрез
        {

            foreach (var item in edgesList)
            {
                item.partitioned = false;

            }
            var q1 = from item in edgesList
                     from itemA in listA
                     from itemB in listB
                     where (itemA.number == item.first_vertex.number
                     && itemB.number == item.second_vertex.number)
                     || (itemB.number == item.first_vertex.number
                     && itemA.number == item.second_vertex.number)
                     select item;
            foreach (var item in q1)
            {
                item.partitioned = true;
```

```
            }

        }

        private int SumArea(List<FMVertex> vertices)//Сумарный вес вершин компоненты
        {
            int sum = 0;
            foreach (var element in vertices)
            {
                sum += element.area;
            }
            return sum;
        }

        public bool BalanceCriterion(List<FMVertex> vertices)
        {

            bool flag = false;
            int count = vertices.Count;
            int commonCount = OutVerticesA.Count + OutVerticesB.Count;


            if (commonCount < 5)
            {
                if (count > 0 && commonCount - count > 0)
                {
                    flag = true;
                }
            }
            else if (commonCount < 10)
            {
                if ((count >= (commonCount / 2) - 1 && count <= (commonCount / 2) +
1) && ((commonCount - count >= (commonCount / 2) - 1 && commonCount - count <=
(commonCount / 2) + 1)))
                {
                    flag = true;
                }
            }
            else
            {
                if ((count >= (commonCount / 2) - commonCount / 8 && count <=
(commonCount / 2) + commonCount / 8) && ((commonCount - count >= (commonCount / 2) -
commonCount / 8 && commonCount - count <= (commonCount / 2) + commonCount / 8)))
                {
                    flag = true;
                }
            }

            return flag;
        }

        private void Gains(List<Edge> edgesList, List<FMVertex> InVertA,
List<FMVertex> InVertB)//расчет стоимости переноса вершины
        {
            foreach (var element in InVertA)//расчет стоимости переноса вершин 1-ой
компоненты
            {
                element.growth = FS(edgesList, element, InVertA) - TE(edgesList,
element);
            }
```

```
            for (int i = 0; i < InVertB.Count; i++)//расчет стоимости переноса вершин
2-ой компоненты
            {
                var element = InVertB[i];
                element.growth = FS(edgesList, element, InVertB) - TE(edgesList,
element);
            }
        }

        private int TE(List<Edge> edgesList, FMVertex vertex)//сила противодействия
        {
            int count = 0;

            foreach (var element in edgesList)
            {
                if (((element.first_vertex.number == vertex.number) ||
(element.second_vertex.number == vertex.number)) && element.partitioned == false)
                {
                    count+=element.weight;
                }
            }

            return count;
        }

        private int FS(List<Edge> edgesList, FMVertex vertex, List<FMVertex>
vertexlist)//дивижущая сила
        {
            int count = 0;
            //проходимось по усім ребрам
            foreach (var element_edge in edgesList)
            {
                //якщо вершина належить розрізаному ребру
                if (((vertex.number == element_edge.first_vertex.number) ||
(vertex.number == element_edge.second_vertex.number)) && element_edge.partitioned ==
true)
                {
                    int edge_count = 0;

                    //пройдемося по усім вершинам переданого списку
                    foreach (var element_vertex in vertexlist)
                    {
                        if ((element_vertex.number ==
element_edge.first_vertex.number) || (element_vertex.number ==
element_edge.second_vertex.number))
                        {
                            edge_count++;
                        }
                    }

                    //якщо  розрізаному ребру належать вершини, що знаходяться в
різних компонентах то "дивижущая сила" збільшується на 1
                    if (edge_count == 1)
                    {
                        count+=element_edge.weight;
                    }
                }
            }

            return count;
        }
```

```
        private FMVertex MaxGains(List<FMVertex> vertexListA, List<FMVertex>
vertexListB, int sum, List<FMVertex> neighbor, int step)//определение вершини,
которую вигоднее всего передвигать
        {
            List<FMVertex> AllVerticesList = new List<FMVertex>();
            List<FMVertex> CopyAllVerticesList = new List<FMVertex>();
            List<FMVertex> MaxGainsList = new List<FMVertex>();
            List<FMVertex> CopyMaxGainsList = new List<FMVertex>();
            List<FMVertex> copyVertexListA = new List<FMVertex>(vertexListA.Clone());
            List<FMVertex> copyVertexListB = new List<FMVertex>(vertexListB.Clone());

            if(step==1)
            {
                AllVerticesList.AddRange(vertexListA.Clone());
                AllVerticesList.AddRange(vertexListB.Clone());
                CopyAllVerticesList.AddRange(AllVerticesList.Clone());
            }
            else if (neighbor.Count>0)
            {
                AllVerticesList.AddRange(neighbor.Clone());
                CopyAllVerticesList.AddRange(AllVerticesList.Clone());
            }

            foreach (var vertex in CopyAllVerticesList)
            {
                if (vertex.fix==true)
                {
                    AllVerticesList.RemoveAll(e=>e.number==vertex.number);
                }
            }


            //сортировка списка вершин по стоимости
            AllVerticesList.Sort((v1, v2) => v1.growth.CompareTo(v2.growth));
            AllVerticesList.Reverse();

            //отбор вершин с наибольшей стоимостью(если вершин с наибольшей
стоимостью несколько, сформируется список)
            // проводим отбор с учетом критерия баланса
            foreach (var element in AllVerticesList)
            {
                //переносим вершину в противоположную компоненту
                Transfer(copyVertexListA, copyVertexListB, element);

                //сохраняем сумарный вес вершин компоненты А после перемещения
                element.balance = SumArea(copyVertexListA);

                //и проверяем исполняется ли критерий баланса
                if (BalanceCriterion(copyVertexListA))
                {
                    //если да добавляем ее в новий список потенциальных вершин
                    MaxGainsList.Add(element);
                }
            }
            if (MaxGainsList.Count>1)
            {
                int i = 0;
                do
                {
```

```
                    CopyMaxGainsList.Add(new FMVertex(MaxGainsList[i].number,
MaxGainsList[i].area, MaxGainsList[i].growth, MaxGainsList[i].fix));
                    i++;
                } while (i<MaxGainsList.Count() && MaxGainsList[i].growth ==
MaxGainsList[i - 1].growth);

                //если там до сих пор осталось больше одной вершины,
                if (CopyMaxGainsList.Count > 1)
                {
                    CopyMaxGainsList.Sort((v1, v2) =>
v1.balance.CompareTo(v2.balance));
                }
            }
            else
            {
                CopyMaxGainsList.AddRange(MaxGainsList.Clone());
            }

            //сортировка по возрастанию возвращаем Last()
            return CopyMaxGainsList.LastOrDefault();
        }

        private void Transfer(List<FMVertex> firstList, List<FMVertex> secondList,
FMVertex vertex)//перенос вершины в противоположную компоненту
        {
            List<FMVertex> copyFirstList = new List<FMVertex>(firstList.Clone());
            List<FMVertex> copySecondList = new List<FMVertex>(secondList.Clone());
            foreach (var item in copyFirstList)
            {
                if (item.number == vertex.number)
                {
                    firstList.RemoveAll(e=>e.number==item.number);
                    secondList.Add(vertex);
                }
            }
            foreach (var item in copySecondList)
            {
                if (item.number == vertex.number)
                {
                    secondList.RemoveAll(e=>e.number==item.number);
                    firstList.Add(vertex);
                }
            }



        }

        private void Unfixing(List<FMVertex> firstList, List<FMVertex>
secondList)//Расфиксация вершин
        {
            foreach (var item in firstList)
            {
                item.fix = false;
            }
            foreach (var item in secondList)
            {
                item.fix = false;
            }
        }
```

```
        private bool Fixed(List<FMVertex> List)//Проверка на фиксацию
        {
            foreach (var item in List)
            {
                if (item.fix == false)
                {
                    return false;
                }
            }
            return true;
        }

        public static int SumCut(List<Edge> edgesList)//Подсчет веса разреза
        {
            int count = 0;
            //проходимось по усім ребрам
            foreach (var element_edge in edgesList)
            {

                if (element_edge.partitioned == true)
                {
                    count += element_edge.weight;
                }
            }
            return count;
        }

        //Вывод на консоль
        //public void OutputGraph(List<Edge> Edges)
        //{
        //    Console.WriteLine("-I--II---Weight--Partitioned");
        //    foreach (var item in Edges)
        //    {
        //        Console.WriteLine(item.ToString());
        //    }
        //}
    }
}
```