Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

**Паралельні та розподілені обчислення**

**Лабораторна робота №7**

Виконав:

Студент групи ПМі-33

Тимчишин Ярема

Львів 2023

**Тема:** Алгоритм Прима.

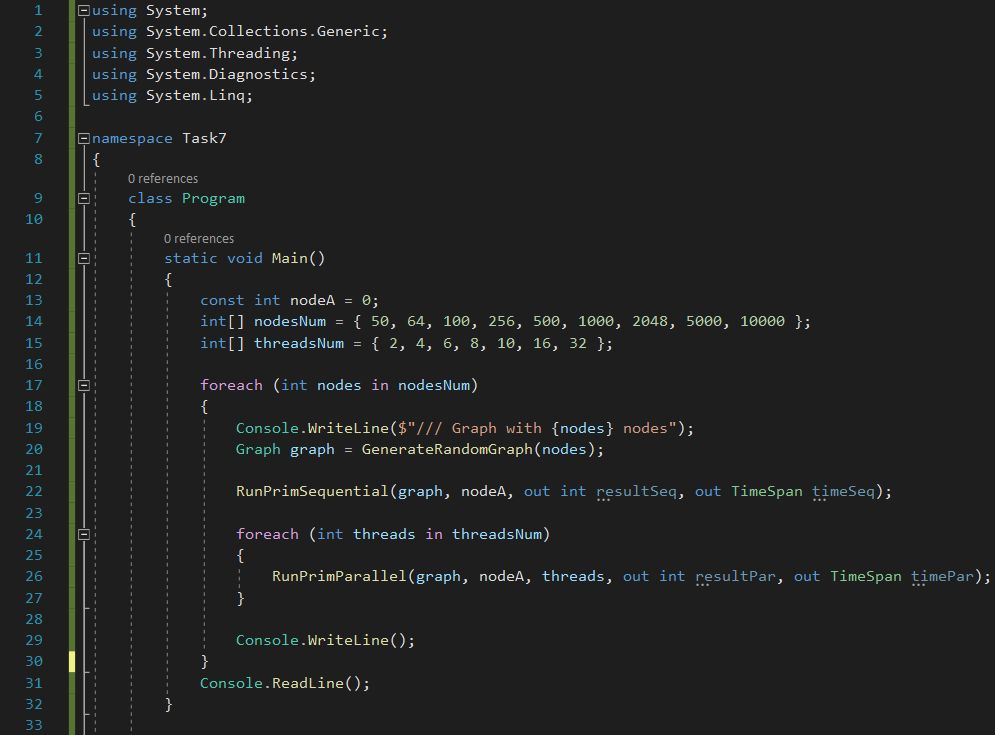
**Завдання:** Для зваженого зв’язного неорієнтованого графа G, використовуючи алгоритм Прима, з довільно заданої вершини a побудувати мінімальне кісткове дерево.

Для різної розмірності графів та довільного вузла a порахувати час виконання програми без потоків та при заданих k потоках розпаралелення.

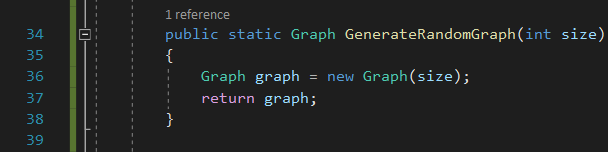
**Хід роботи**

Використовуючи мову програмування C# написав програму для побудови мінімального кісткового дерева з довільного вузла за допомогою розпаралелення методу Прима.

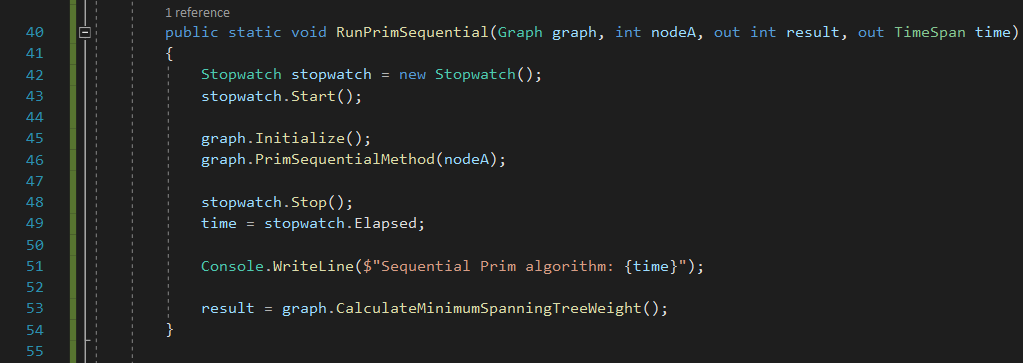
1. Метод **Main()**. Оголошення всіх змінних, виклик методу для генерації графу та виклик двох основних методів RunPrimSequential() та RunPrimParallel().



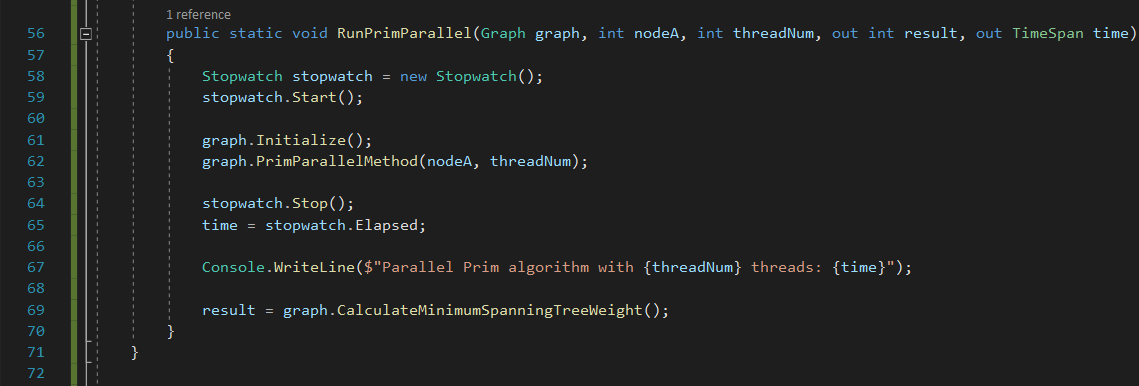
1. Метод **GenerateRandomGraph()**. Цей метод отримує кількість вершин, яка має бути у графі, після чого створює екземпляр класу Graph, який відповідатиме за представлення та роботу з графом, і потім повертає цей граф.



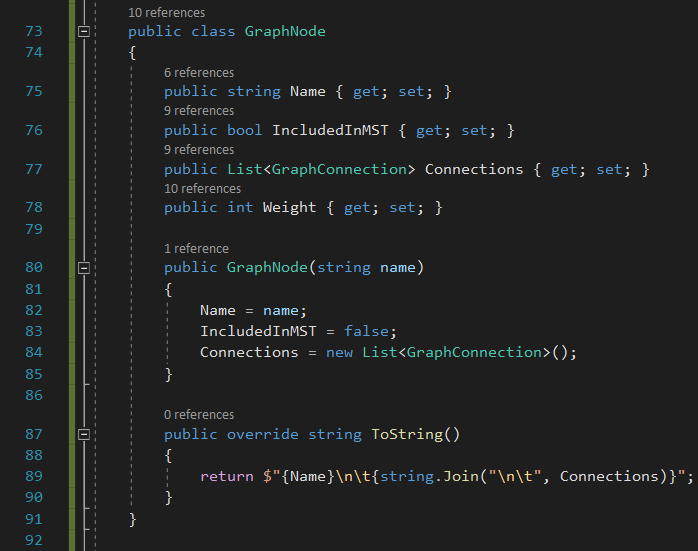
1. Метод **RunPrimSequential()**. Цей метод викликає послідовний алгоритм Прима для заданого графа та для заданої вершини і виводить час виконання алгоритму.



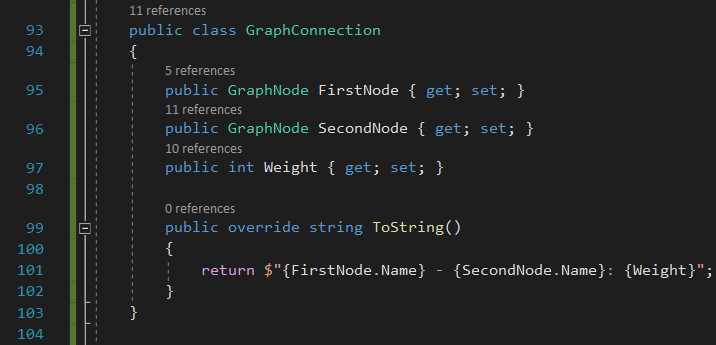
1. Метод **RunPrimParallel()**. Цей метод викликає паралельний алгоритм Прима з різною кількістю потоків для заданого графа та для заданої вершини і виводить час виконання алгоритму з різною кількістю потоків.



1. Клас **GraphNode**. Цей клас представляє вузол у графі. Детальніше розглянемо властивості та методи цього класу:
   * Властивість **Name**. Ця властивість представляє ім'я або ідентифікатор вершини графа.
   * Властивість **Passed**. Ця властивість є булевим прапорцем, який використовується в алгоритмах обходу графа для позначення того, чи була вершина відвідана під час обходу.
   * Властивість **Connections**. Ця властивість є списком об'єктів GraphConnection, що представляють зв'язки або ребра, які з'єднують цю вершину з іншими вершинами графа.
   * Властивість **Weight**. Ця властивість представляє вагу, пов'язану з вершиною.
   * Конструктор **GraphNode**. Цей конструктор приймає параметр name та ініціалізує властивості об'єкта GraphNode. Він встановлює надане ім'я у Name, ініціалізує Passed у false і створює порожній список для Connections.
   * Метод **ToString()**. Цей метод надає користувацьке рядкове представлення об'єкта GraphNode. Він повертає рядок, що містить ім'я вузла та відформатований список його з'єднань.



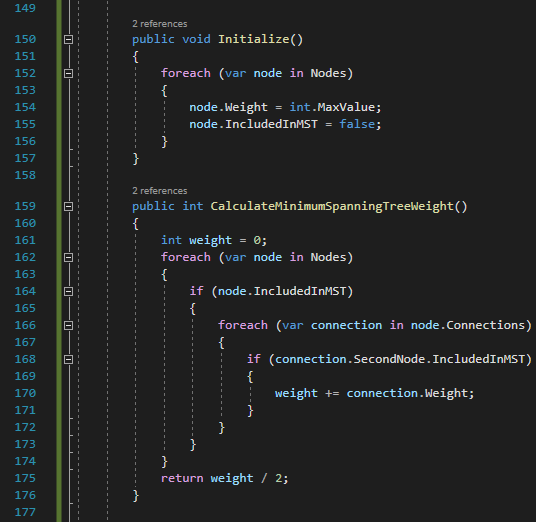
1. Клас **GraphConnection**. Цей клас представляє зв'язок або ребро між двома вершинами графа. Детальніше розглянемо властивості та методи цього класу:
   * Властивість **FirstNode**. Ця властивість представляє перший із з'єднаних вузлів.
   * Властивість **SecondNode**. Ця властивість представляє другий підключений вузол.
   * Властивість **Weight**. Ця властивість представляє вартість проходження ребра між двома вузлами.
   * Метод **ToString()**. Цей метод надає користувацьке рядкове представлення об'єкта GraphConnection. Він повертає рядок, який містить імена першої та другої вершин, з'єднаних цим ребром, а також вагу, пов'язану зі з'єднанням.



1. Клас **Graph**. Цей клас представляє структуру даних графа і містить методи для ініціалізації графа, побудови мінімального кісткового дерева за алгоритмом Прима, а також виконання цієї операції як послідовно, так і паралельно. Детальніше розглянемо властивості та методи цього класу:
   * Властивість **Nodes**. Ця властивість є списком, який містить екземпляри GraphNode, що представляють вузли графу.
   * Властивість **Connections**. Ця властивість є списком, що містить екземпляри GraphConnection, які представляють зв'язки або ребра між вершинами графа.
   * Конструктор **Graph**. Цей конструктор ініціалізує новий об'єкт Graph. Він приймає ціле число nodeNum як параметр, який визначає кількість вершин у графі. Всередині конструктора генерується випадковий граф з вказаною кількістю вершин. Граф будується шляхом створення вершин і випадкових зв'язків між ними.



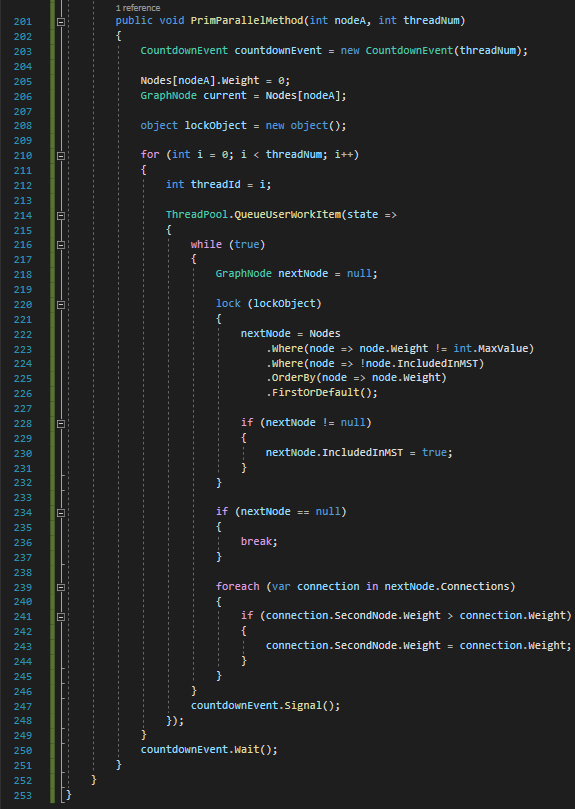
* + Метод **Initialize()**. Цей метод відповідає за ініціалізацію властивості Weight кожної вершини графа максимальним значенням. Мета цього методу - підготувати граф для алгоритму пошуку найкоротших шляхів між вершинами. Встановивши для всіх ваг вершин максимальне значення, алгоритм може почати роботу з чистого аркуша, вважаючи всі вершини невідвіданими, а відстані - невідомими.
* Метод **CalculateMinimumSpanningTreeWeight()**. Цей метод обчислює загальну вагу мінімального кісткового дерева шляхом підсумовування ваг усіх ребер у дереві, беручи до уваги те, що кожне ребро рахується двічі (по одному разу для кожної з двох його кінцевих точок), і тому загальну вагу потрібно розділити на 2, щоб отримати правильну вагу MST.



* Метод **PrimSequentialMethod()**. Цей метод послідовно застосовує алгоритм Прима для пошуку мінімального кісткового дерева від заданої початкової вершини. Він перебирає всі з'єднання (ребра) поточної вершини. Для кожного з'єднання перевіряється, чи інший кінець з'єднання ще не включено до MST і чи вага з'єднання менша за вагу іншого кінця. Якщо обидві умови виконано, він оновлює вагу іншого кінця до ваги з'єднання, фактично додаючи це ребро до MST. Після чого цей метод оновлює поточну вершину до наступної вершини, яка ще не включена до MST і має найменшу вагу серед цих вершин. Цей крок є ключовою частиною алгоритму Прима і допомагає знайти наступний вузол для включення в MST на основі ребра з найменшою вагою.



* Метод **PrimParallelMethod()**. Цей метод застосовує алгоритм Прима з використанням заданої кількості потоків (threadNum). Він використовує пул потоків для паралельного виконання алгоритму, оновлюючи ваги вузлів та обробляючи синхронізацію між потоками за допомогою події CountdownEvent.



Результат роботи програми

