**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ  
  
  
  
ПОЯСНЮВАЛЬНА ЗАПИСКА**

**ДО КУРСОВОЇ РОБОТИ З ПРОЕКТУВАННЯ АЛГОРИТМІВ**

**ТА ПРОГРАМУВАННЯ**

на тему:

«Алгоритм Прима»

Виконала студентка 2 курсу  
 групи КН-21

*Тракало Анастасія Андріївна*

Засвідчую, що в курсовій роботі немає  
 запозичень з праць інших авторів  
 без відповідних посилань

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник курсової роботи:   
 *кандидат технічних наук, доцент*

*Мінаєва Юлія Іванівна*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Оцінка за курсову роботу: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2023

**Київський національний університет імені Тараса Шевченка**

**Факультет інформаційних технологій**

**Кафедра інтелектуальних технологій**

**ЗАВДАННЯ**

на курсову роботу з проектування алгоритмів та програмування

студентці Тракало Анастасії Андріївні

**1. Тема роботи:** «Алгоритм Прима».

**2. Термін здачі закінченого проекту:** «26» травня 2023 р.

**3. Вихідні дані до проекту:** Алгоритм побудови мінімального кістякового дерева зваженого зв’язного неорієнтованого графу, відомий як «Алгоритм Прима».

**4. Зміст роботи:**

1. Аналіз особливостей задачі побудови мінімального кістякового дерева зв’язного неорієнтованого графу. Узагальнений опис алгоритму Прима.

2. Проектні рішення щодо створення програмного застосунку для вирішення задачі побудови мінімального кістякового дерева зваженого зв’язного неорієнтованого графу шляхом застосування алгоритму Прима.

3. Програмна реалізація застосунку, що візуально відображатиме роботу алгоритму Прима.

**5. Перелік презентаційного матеріалу:**

1. Мета, об’єкт та предмет дослідження курсової роботи (1 слайд)

2. Постановка задачі (1-2 слайди)

3. Узагальнений опис алгоритму Прима (1-2 слайди)

4. Проектування програмного застосунку (3-4 слайди)

5. Інтерфейс та результати роботи програмного застосунку (2 слайди)

6. Висновки (1 слайд)

**6. Дата видачі завдання** «1» лютого 2023 р.

Графік виконання курсової роботи

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва етапу** | **Терміни** | **Примітки / відмітка про виконання** |
| 1 | Вибір теми та керівника курсової роботи | 1 – 14  лютого | Заява на виконання курсової роботи, що підписана студентом керівником |
| 2 | Обговорення з керівником постановки завдання та змісту пояснювальної записки до курсової роботи | 14 – 28  лютого | Заповнений бланк завдання на курсову роботу, що підписаний студентом та керівником роботи |
| 3 | Аналіз постановки задачі, формалізація задачі, вибір методів та засобів реалізації поставленої задачі, аналіз літературних джерел | 1 – 12  березня | Сформований матеріал до розділу 1 пояснювальної записки курсової роботи, оформлення списку джерел |
| 4 | *Перше узгодження з керівником* | 13 – 19  березня |  |
| 5 | Розробка алгоритму, вибір структур даних, проектування програмного інтерфейсу з користувачем. | 13 березня  - 1 квітня | Сформований матеріал до розділу 2 пояснювальної |
| 6 | *Друге узгодження з керівником КОНТРОЛЬНА ПЕРЕВІРКА* | 1 – 15 квітня |  |
| 7 | Розробка та тестування програмного продукту. | 1 – 23 квітня | Готовий програмний продукт |
| 8 | *Третє узгодження з керівником. Демонстрація базового варіанту програмного продукту.* | 17 – 23 квітня |  |
| 9 | Доопрацювання програмного продукту, всебічне заключне тестування, розробка тестових прикладів та керівництва користувача. | 17 – 27 квітня | Сформований матеріал до розділу 3 пояснювальної записки, підготовлений демонстраційний приклад роботи програми |
| 10 | Остаточне оформлення пояснювальної записки | 27 – 30 квітня | Готова пояснювальна записка та програмний застосунок |
| 11 | Подання роботи керівнику, перевірка роботи на плагіат, підготовка презентаційного матеріалу | 1 – 14 травня | Готова пояснювальна записка та програмний застосунок, результати перевірки на плагіат |
| 12 | Подання роботи до захисту | 15 – 26 травня | Готова пояснювальна записка, програмний застосунок, презентація для захисту курсової роботи, демонстраційний приклад роботи програми |

Керівник роботи \_\_\_\_\_\_\_\_\_ Мінаєва Ю. І.

Завдання прийняла до виконання  Тракало А. А.

**ЗМІСТ**

[ВСТУП 4](#_Toc134974580)

[РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД АЛГОРИТМУ ПРИМА 5](#_Toc134974581)

[1.1 Огляд алгоритмів пошуку мінімального кістякового дерева 5](#_Toc134974582)

[1.2 Узагальнений алгоритм розв'язку задачі 5](#_Toc134974583)

[1.3 Передумови для застосування алгоритму Прима 7](#_Toc134974584)

[1.4 Перелік задач, до яких застосовується алгоритм Прима 7](#_Toc134974585)

[1.5 Переваги та недоліки 8](#_Toc134974586)

[1.6 Постановка задачі 9](#_Toc134974587)

[Висновки до розділу 1 10](#_Toc134974588)

[РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ 11](#_Toc134974589)

[2.1 Опис методу та алгоритму розв’язку поставленої задачі 11](#_Toc134974590)

[2.2 Вибір мови програмування та середовища 12](#_Toc134974591)

[2.3 Вибір структур даних 13](#_Toc134974592)

[2.4 Діаграма варіантів використання 14](#_Toc134974593)

[2.5 Діаграма послідовності 15](#_Toc134974594)

[Висновки до розділу 2 15](#_Toc134974595)

[РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ 16](#_Toc134974596)

[3.1 Діаграма класів 16](#_Toc134974597)

[3.2 Керівництво користувача програмного продукту 16](#_Toc134974598)

[3.3 Опис тестових випадків та перевірка працездатності застосунку 20](#_Toc134974599)

[Висновки до розділу 3 24](#_Toc134974600)

[ВИСНОВКИ 25](#_Toc134974601)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 26](#_Toc134974602)

[ДОДАТКИ 27](#_Toc134974603)

## ВСТУП

**Мета роботи:** систематизувати отримані теоретичні знання з дисциплін «Проектування та аналіз алгоритмів», «Дискретні структури», «Алгоритмізація та програмування», «Об’єктно-орієнтоване програмування», «Комп’ютерна графіка», «Технологія створення програмних продуктів»; закріпити навички самостійної роботи та поглибити практичні навички зі створення працездатного програмного застосунку, який ефективно та повноцінно вирішує поставлене завдання, а саме: реалізує алгоритм Прима, який є жадібним алгоритмом побудови мінімального кістякового дерева зваженого зв’язного неорієнтованого графу.

**Опис структури роботи за розділами:** курсова робота поділяється на три розділи. Перший розділ містить загальний аналітичний огляд алгоритму Прима, сфери його застосування, переваги й недоліки, а також функціональні й нефункціональні вимоги до застосунку; другий присвячений проєктуванню програмного продукту з деталізованим описом алгоритму розв’язку задачі та обґрунтуванням структур даних; третій розділ містить реалізацію програмного продукту, тестування його працездатності, результати роботи й аналіз недоліків.

**Використані засоби розробки:** Qt Creator.

**Практичне значення одержаних результатів:** алгоритм Прима можна застосувати для побудови схеми доріг між населеними пунктами з мінімальними затратами ресурсів, які закуплятимуться для побудови доріг; для визначення мінімальної довжини кабелю необхідного для постачання Інтернету до осель певного населеного пункту тощо.

## РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД АЛГОРИТМУ ПРИМА

### 1.1 Огляд алгоритмів пошуку мінімального кістякового дерева

Одним із важливих завдань теорії графів є знаходження мінімального кістякового дерева. Мінімальне кістякове дерево - це підграф графу, який є деревом, що містить всі вершини графу та не має циклів, із найменшою вагою серед всіх можливих кістякових дерев графу. Задача знаходження мінімального кістякового дерева є складовою багатьох практичних задач: наприклад, алгоритм застосовується в таких сферах як мережеве планування, транспортна логістика, телекомунікації тощо.

### 1.2 Узагальнений алгоритм розв'язку задачі

Алгоритм Прима – один з найефективніших алгоритмів знаходження мінімального кістякового дерева. Він належить до категорії жадібних алгоритмів, роботу якого можна поділити на дві фази. У першій фазі обирається будь-яка вершина графу та додається до множини вершин, які входять до мінімального кістякового дерева. У другій фазі ми починаємо з дерева, що складається з однієї вершини, і на кожному кроці знаходимо ребро з найменшою вагою серед усіх ребер, які з'єднують множину вершин у дереві з іншими вершинами у графі. Цей процес триває до тих пір, поки всі вершини графу не ввійдуть до мінімального кістякового дерева.

Узагальнена схема алгоритму має такий вигляд:

1. На вхід подається зважений зв’язний неорієнтований граф. Граф можна подати у вигляді матриці суміжності або списку суміжності.
2. Довільно обирається вершина, з якої починається пошук мінімального кістякового дерева (далі – МКД), і додається до множини вершин, які входять в МКД, та до черги з пріоритетом.
3. Усі ребра, які з’єднують вершину, що входить до МКД, та вершини, які ще не входять в МКД, позначаються як потенційні ребра для додавання до МКД.
4. У чергу з пріоритетом додаються всі потенційні ребра, де пріоритетом для сортування є вага ребер.
5. З черги з пріоритетом вибирається ребро з найменшою вагою та перевіряється, чи воно з’єднує дві різні компоненти графу.
6. Якщо ребро з’єднує вершину, яка належить до МКД, з вершиною, яка ще не належить до МКД, то ребро вилучається з черги з пріоритетом і додається до МКД.
7. Кроки 3-6 повторюються доти, доки всі вершини графа не будуть включені до МКД. По завершенні алгоритму черга з пріоритетом буде порожньою.

Приклад роботи такого алгоритму, який наведено у [1], подано на рисунку 1.1:

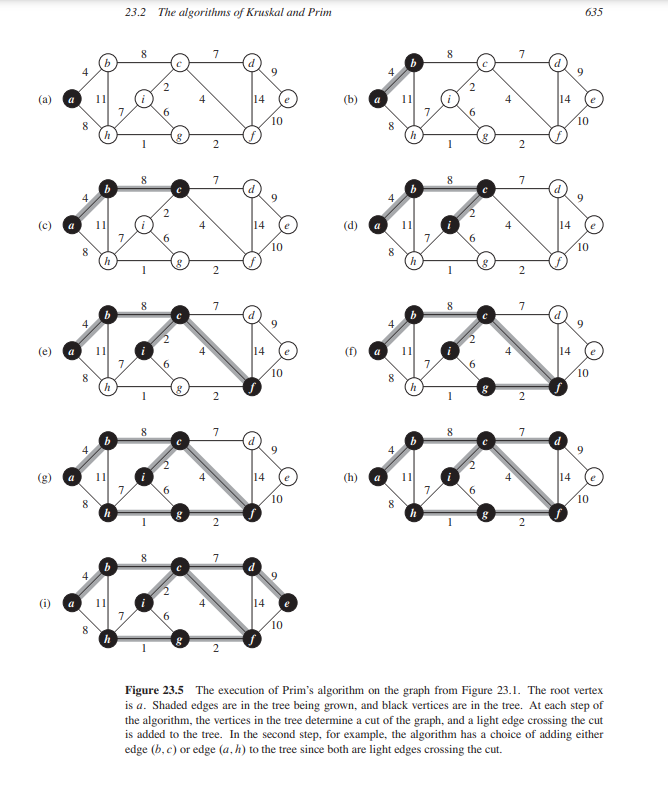


Рисунок 1.1 – Приклад знаходження мінімального кістякового дерева

На рисунку 1.1 початковою вершиною є вершина *а.* Затінені сірим – ребра, що додаються до мінімального кістякового дерева, а чорним позначено вершини, які включено до МКД. На кожному кроці алгоритму до МКД додається ребро з найменшою вагою. Так, наприклад, на кроці *(b)* алгоритм може обрати додавання до дерева або ребра *(b, c)*, або ребра *(a, h)*, оскільки обидва мають найменшу однакову вагу.

### 1.3 Передумови для застосування алгоритму Прима

Алгоритм Прима має деякі умови для його застосування. По-перше, граф повинен бути зв'язним та неорієнтованим. Зв'язність графу означає, що між будь-якими двома вершинами графу існує шлях. Неорієнтований граф – це граф, що містить лише ребра та не містить дуг.

По-друге, граф повинен бути простим, тобто не містити кратних (паралельних) ребер та петель. Кратними називаються ребра, що сполучають одну й ту ж пару вершин. Петлею називають ребро, що сполучає вершину саму із собою.

По-третє, для застосування Алгоритму Прима необхідно мати можливість визначити вагу кожного ребра графа. Якщо вагові коефіцієнти ребер графа не задані або визначені неоднозначно, то потрібно спочатку провести попередню обробку графу та встановити фіксовану вагу ребер.

### 1.4 Перелік задач, до яких застосовується алгоритм Прима

Алгоритм Прима застосовується у доволі широкому спектрі галузей. Наприклад, алгоритм можна використовувати при побудові мереж зв’язку для забезпечення мінімальних витрат; в транспортній логістиці для оптимізації маршрутів доставки товарів та зменшення часу на транспортування; в електроенергетиці для знаходження мінімальної кількості кабелів, щоб підключити джерела електроенергії в мережі; у біоінформатиці для знаходження філогенетичних відносин між організмами. Також можна застосовувати алгоритм для виявлення плагіату в програмному коді, для розміщення мінімальної кількості релейних вузлів в оптичних мережах, для кластеризації даних, для вирішення задачі комівояжера з використанням мінімального кістякового дерева тощо. Загалом алгоритм Прима дозволяє покращити ефективність та економічність різних систем та процесів.

### 1.5 Переваги та недоліки

Виділимо основні переваги й недоліки алгоритму Прима.

До переваг алгоритму Прима належать:

* ефективність: алгоритм Прима є ефективним алгоритмом, який працює за час, пропорційний O(E\*log(V)), де V - кількість вершин, а E - кількість ребер в графі;
* оптимальність: алгоритм Прима знаходить мінімальне кістякове дерево з найменшою вагою;
* простота реалізації: алгоритм Прима досить простий в реалізації та застосовний в більшості мов програмування;
* можливість імплементування в життя: алгоритм Прима можна застосовувати на практиці для рішення багатьох реальних задач, таких як маршрутизація в комп'ютерних мережах, планування маршрутів для автомобілів або літаків, аналіз геномних даних тощо;
* робота з невпорядкованими графами: алгоритм Прима може бути використаний для роботи з невпорядкованими графами, у яких порядок ребер не визначений. Як результат, алгоритм стає особливо корисним та прикладним для задач, пов'язаних з даними, які не мають структури або не є лінійними.

Недоліки алгоритму:

* чутливість до зміни ваги ребер. Якщо вага ребра змінюється, то потрібно перерахувати вагу всього дерева. Це досить затратно в плані часу, особливо якщо граф великий;
* не завжди знаходить найкоротший шлях. Алгоритм Прима знаходить мінімальне кістякове дерево, а не найкоротший шлях між двома вершинами. Якщо в графі є дві вершини, які знаходяться далеко одна від одної, то шлях між ними, знайдений за допомогою алгоритму Прима, може бути довгим;
* не працює з не зв'язними графами. Якщо граф не зв'язний, тобто складається з кількох компонентів, то алгоритм Прима не зможе знайти мінімальне кістякове дерево для всього графа. Для кожної компоненти графа потрібно запустити алгоритм окремо.

### 1.6 Постановка задачі

Основна ідея програмного продукту – реалізація алгоритму Прима. Необхідно забезпечити введення вхідних даних – зваженого зв’язного неорієнтованого графа, та завантаження вихідних даних – мінімального кістякового дерева. Опис проектування програмного продукту з обґрунтуванням вибору мови програмування та структур даних наведено в другому розділі.

Якщо розглядати алгоритм у вигляді «чорної скрині», то на вхід ми подаємо зважений зв’язний неорієнтований граф через список суміжності, а на виході отримаємо мінімальне кістякове дерево. «Чорну скриню» зображено на рисунку 1.2:



Рисунок 1.2 – Алгоритм Прима у вигляді «чорної скрині»

Сформуємо ключові функціональні та нефункціональні вимоги до програмного продукту.

Функціональні вимоги до програмного застосунку полягають у наступному:

* програма повинна забезпечувати можливість зчитування графу з текстового файлу. Граф вводитиметься у вигляді списку суміжності. Перший рядок текстового файлу має містити кількість вершин у графі. Кожен наступний рядок містить інформацію про сусідні вершини для певної вершини графа. Кожен рядок повинен починатися з індексу вершини, а потім за ним слідують пари чисел, які відповідають індексам сусідніх вершин та вагам ребер, що їх з'єднують з поточною вершиною. Кожна пара чисел повинна бути розділена пробілом;
* програма повинна виконувати алгоритм Прима для обчислення мінімального кістякового дерева графу;
* можливість візуалізації мінімального кістякового дерева у графічному вигляді на екрані;
* можливість перевіряти граф на наявність кратних ребер та петель, оскільки алгоритм Прима працює тільки з простими графами;
* можливість завантаження отриманого мінімального кістякового дерева.

Нефункціональні вимоги застосунку включають:

* ефективність роботи;
* запобігання виникненню помилок;
* інтуїтивність, зрозумілість та зручність використання для користувача.

### Висновки до розділу 1

Основним завданням є розробка програмного продукту з реалізацією алгоритму Прима для побудови мінімального кістякового дерева. Застосунок повинен передбачити введення зваженого зв’язного неорієнтованого графу з текстового файлу, перевірку графу на наявність кратних ребер та петель, візуалізацію мінімального кістякового дерева на екрані та можливістю завантаження отриманого дерева. Також необхідно забезпечити кілька різних тестових прикладів для тестування роботи програмного продукту. Детальний опис проектування застосунку, вибір мови програмування та фреймворку наведено в другому розділі.

## РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ

### 2.1 Опис методу та алгоритму розв’язку поставленої задачі

Алгоритм Прима можна зобразити у вигляді блок-схеми, яка зображена на рисунку 2.1.

Алгоритм поділяється на такі етапи:

1. Ініціалізуємо ключі всіх вершин як нескінченні та батька кожної вершини як -1.
2. Створюємо порожню чергу з пріоритетом. Кожен елементи черги – пара (вага ребра, вершина). Вага – ключ, що використовується як перший елемент пари, так як перший елемент за замовчуванням використовується для порівняння двох пар.
3. Ініціалізуємо всі вершини як ті, що поки не входять до МКД. Вони зберігатимуться в булевому масиві, який необхідний, щоб вершина, яка вже врахована, не була включена до черги з пріоритетом знову.
4. Вставляємо вершину, з якої починатимемо пошук, у чергу з пріоритетом та встановлюємо значення її ключа як 0.
5. Далі повторюватимемо цикл, доки черга з пріоритетом не буде порожньою. Вибираємо вершину з найменшим ключем з черги з пріоритетом. Нехай вона буде u. Включаємо вершину до МКД, використовуючи булевий масив і позначаючи його значення як true.
6. У циклі проходимося по всіх суміжних вершин до вибраної u і так повторюватимемо для кожної для кожної вершини v:

якщо вага ребра (u, v) менша, ніж ключ v та v ще не включено до МКД, то

* ключ v оновлюється, який тепер дорівнюватиме вазі ребра (u, v);
* v вставляється у чергу з пріоритетом;
* ребро, що включається до МКД, зберігається в масиві батька.

Якщо черга з пріоритетом порожня, алгоритм завершено.

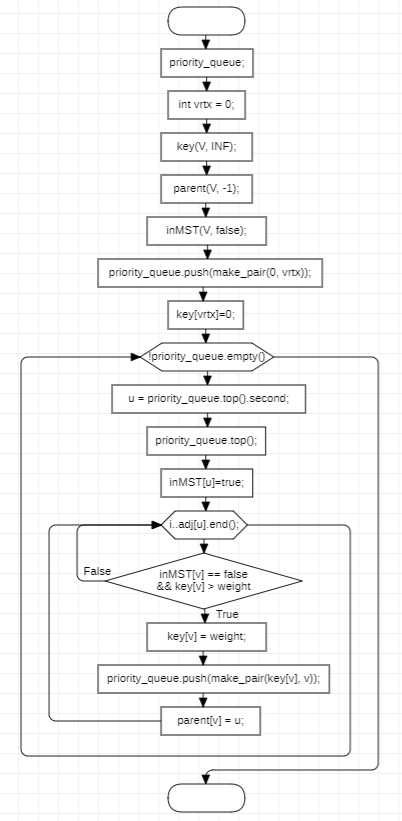


Рисунок 2.1 – Алгоритм знаходження мінімального кістякового дерева

### 2.2 Вибір мови програмування та середовища

Програмний продукт реалізуватимемо мовою програмування C++, оскільки вона має багато бібліотек та фреймворків для розробки програм з графічним інтерфейсом та візуалізацією графів. Бібліотеки в свою чергу містять різноманітні можливості для пристосування та налаштування програми відповідно до вимог. Крім цього, С++ лояльна до масштабування, тобто можна вільно додавати нові функції та класи за потреби, та високопродуктивна мова, що дає переваги в часі та ефективності.

Для реалізації графічного інтерфейсу користувача застосовуватимемо фреймворк ***Qt***. Qt – потужна бібліотека, яка дозволяє створювати користувацький інтерфейс швидко та ефективно й пропонує широкий вибір елементів інтерфейсу, таких як кнопки, поля введення, вікна й ін. Для візуалізації графа та мінімального кістякового дерева використовуватимемо бібліотеку ***Graphviz***, оскільки вона надає різноманітні варіанти представлення графів та підтримує їх експорт у різних форматах, як PNG, SVG, PDF тощо.

Середовище розробки програмного застосунку – ***Qt Creator***.

### 2.3 Вибір структур даних

У застосунку будуть реалізовані такі класи: Graph, MST, Main.

Клас **Graph** служить для представлення графу через список суміжності та для роботи з ним. Клас міститиме методи: readGraphFromFile - для зчитування графа з файлу, hasParallelEdges - для перевірки графу на наявність в ньому кратних ребер та петель, drawGraph - для побудови та візуалізації структури графу.

Клас **MST** необхідний для реалізації алгоритму Прима. Клас міститиме методи: getMST - для знаходження мінімального кістякового дерева за допомогою алгоритму Прима, drawMST – для побудови та візуалізації МКД, saveToFile - для збереження знайденого МКД у файл.

Для реалізації алгоритму Прима використовуватиметься структура даних – черга з пріоритетом, що доступна у бібліотеці STL C++. Вона дозволить зберігати вершини графу, які ще не були включені до МКД, та знаходити вершину з найменшою вагою ребра, яке з'єднує її з вже включеними вершинами. Перевагами використання черги з пріоритетом є ефективність та простота реалізації. По-перше, завдяки тому, що вершини зберігатимуться у відсортованому порядку з найменшою вагою, алгоритм працюватиме швидко, з мінімальними часовими затратами. По-друге, використання STL, яка містить уже готову реалізацію черги з пріоритетом, дозволить значно скоротити час на написання алгоритму Прима та полегшить роботу.

Клас **MainWindow** призначений для взаємодії з користувачем та керування вікнами програми, які матитуть відповідні кнопки та методи для їх обробки. Клас включатиме слоти: openFile – для завантаження файлу з графом; visualizeGraph – для візуального відображення графа на інтерфейсі користувача; saveGraph – для збереження зображення графа; visualizeMST – для відображення МКД на інтерфейсі користувача; saveMST – для збереження дерева.

### 2.4 Діаграма варіантів використання

Відповідно до функціональних вимог до застосунку користувач може виконувати 3 основні функції: вибрати файл з графом для зчитування, побудувати граф з візуалізацією на екрані й можливістю збереження зображення графа, знайти мінімальне кістякове дерево (МКД) з можливістю збереження дерева у файл.

Діаграму варіантів використання представлено на рисунку 2.2:

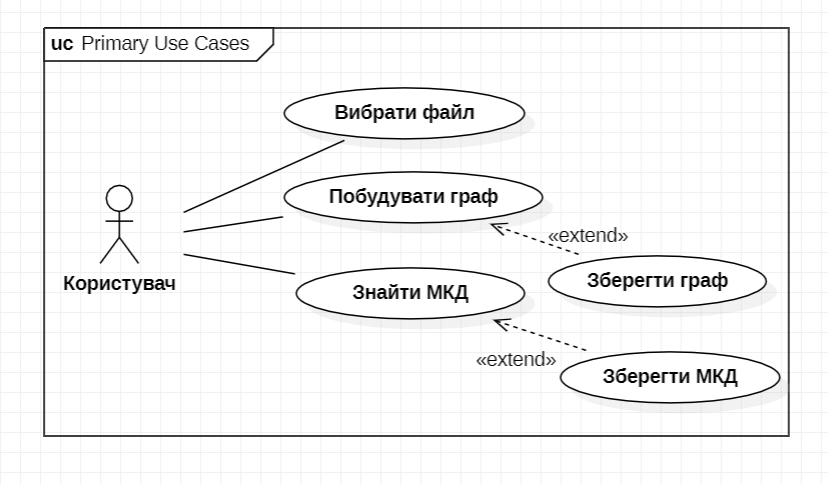


Рисунок 2.2 – Діаграма варіантів використання

### 2.5 Діаграма послідовності

Визначимо основні процеси програми та зобразимо їх на діаграмі послідовності, яку представлено на рисунку 2.3:

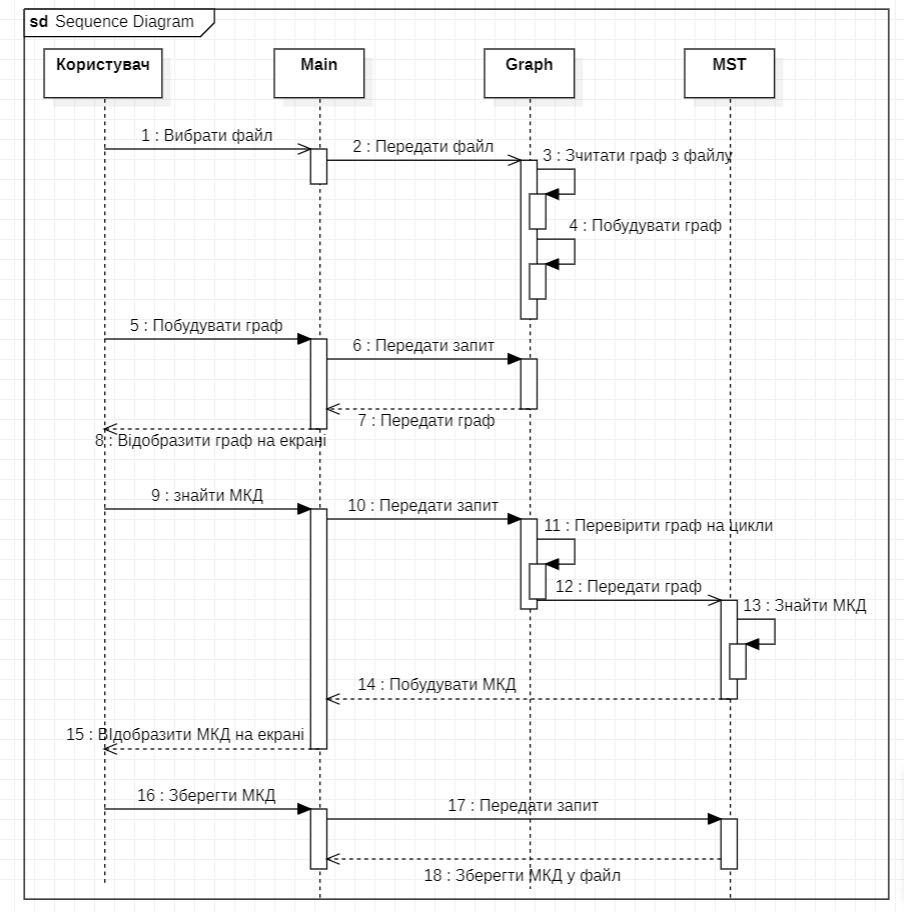


Рисунок 2.3 – Діаграма послідовності

### Висновки до розділу 2

Задача – розробити програмний продукт на мові програмування С++ в середовищі розробки Qt Creator з реалізацією графічного користувацького інтерфейсу за допомогою Qt Designer. Детальний опис реалізації та тестування програмного застосунку наведено в розділі 3.

## РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

### 3.1 Діаграма класів

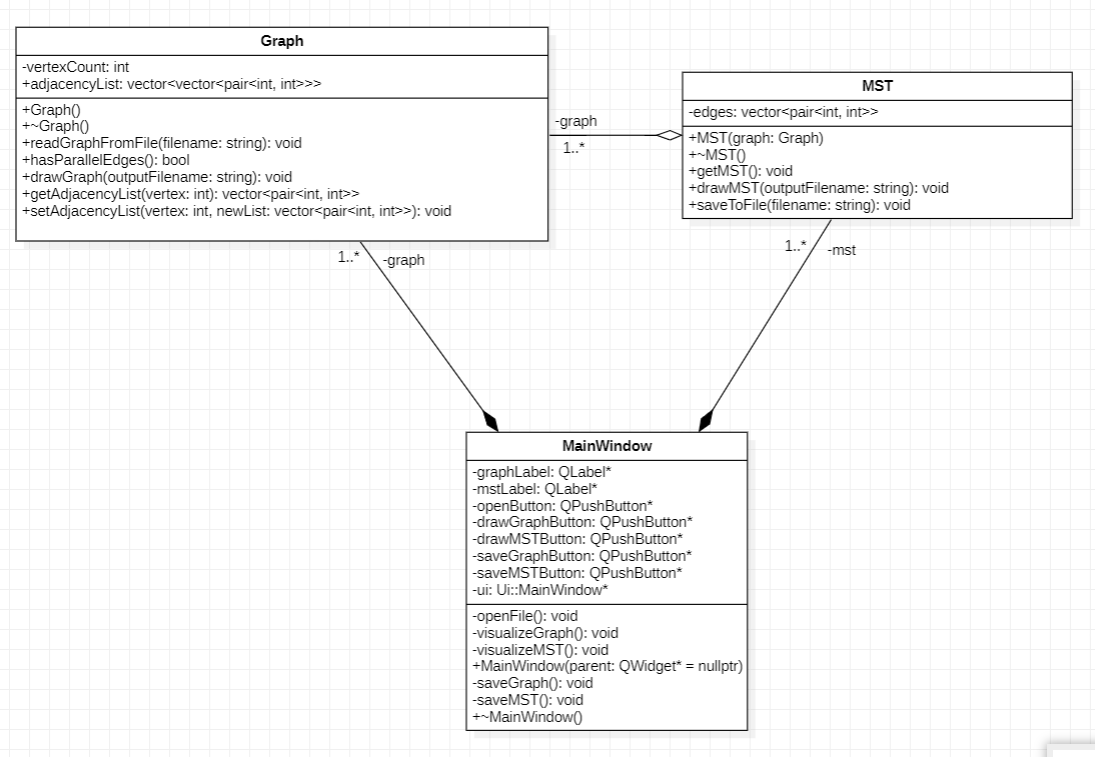


Рисунок 3.1 – Діаграма класів

### 3.2 Керівництво користувача програмного продукту

Для того, щоб використовувати програмний продукт, потрібно, насамперед, завантажити архів із застосунком на свій пристрій. Після розархівування отримуємо наступні файли:

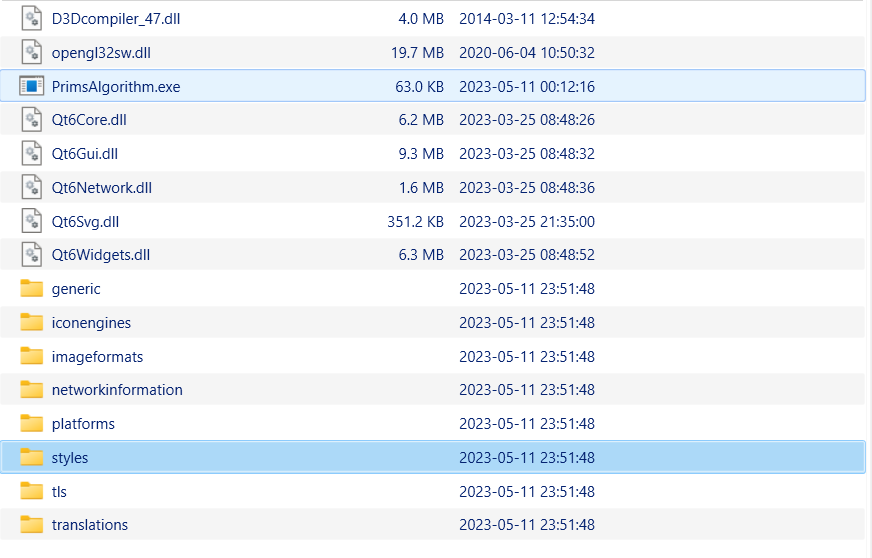


Рисунок 3.2 – Вигляд програмного продукту

Щоб запустити застосунок, натискаємо на файл **PrimsAlgorithm.exe**. Після чого відкриється вікно з графічним інтерфейсом програмного продукту:

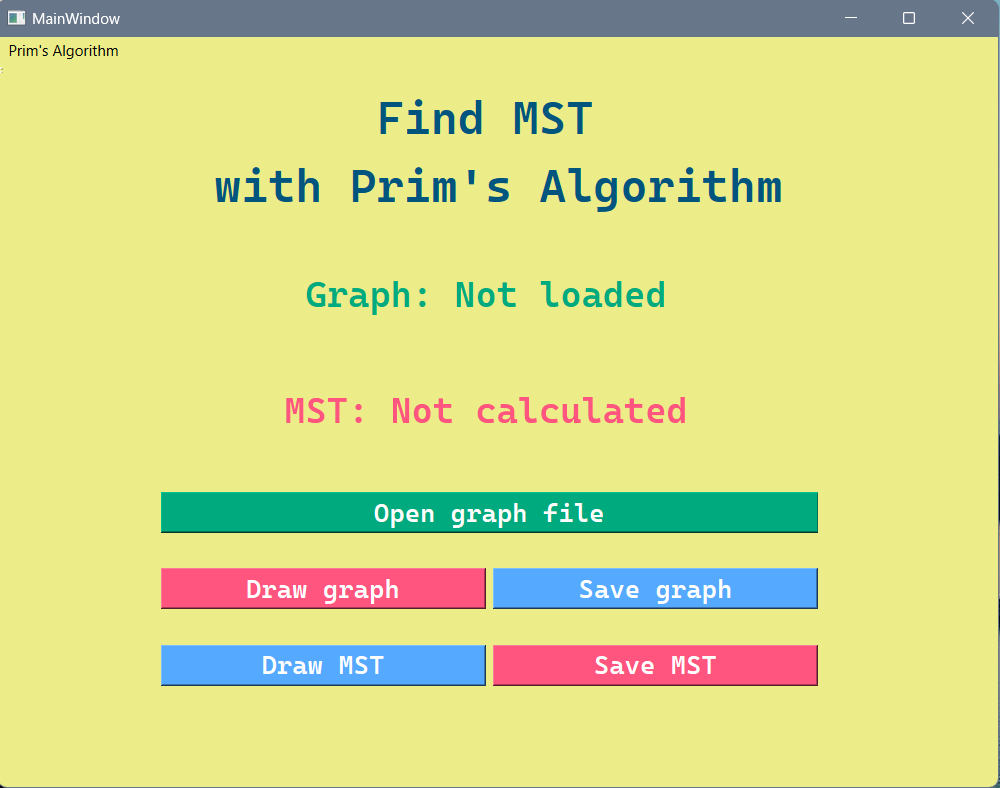


Рисунок 3.3 – Графічний інтерфейс програмного продукту

Натискаємо на кнопку ***Open graph file****,* після чого відкриється вікно для вибору текстового документу з даними графа:

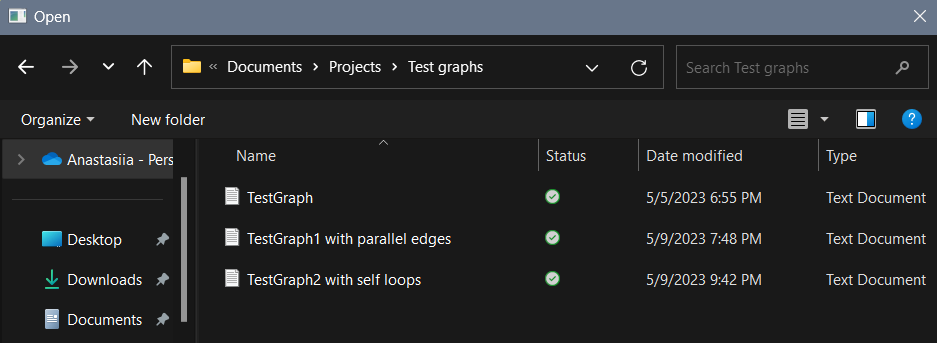


Рисунок 3.4 – Вибір текстового файлу з даними графа

Якщо файл було завантажено коректно, то біля мітки «**Graph:»** повідомлення зміниться на «Loaded».

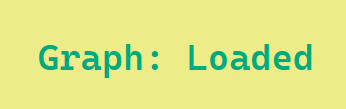


Рисунок 3.5 – Повідомлення про успішне завантаження файлу

Щоб побудувати граф, натискаємо на кнопку ***Draw graph***. Відкриється вікно із зображенням побудованого графу:

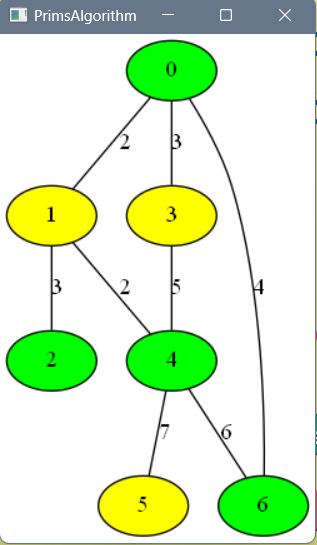


Рисунок 3.6 – Приклад побудованого графу

Щоб зберегти граф, натисніть ***Save graph***. Граф буде збережено як «graph.png».

Для побудови мінімального кістякового дерева за алгоритмом Прима натисніть ***Draw MST***. Відкриється вікно із зображенням побудованого МКД:

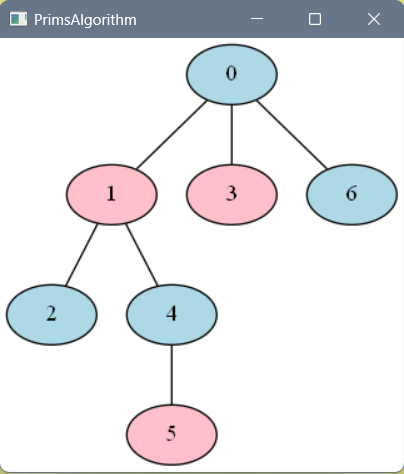


Рисунок 3.7 – Приклад побудованого МКД

Щоб зберегти МКД, натисніть ***Save MST***. Дерево буде збережено як «mst.png» та «mst\_edges.txt» зі списком ребер, які ввійшли до МКД.

Зверніть увагу(!), що алгоритм Прима працює тільки з простими графами, тобто потрібно завантажити файл з графом, який не містить кратних ребер чи петель. У випадку, якщо завантажено не простий граф, при натисканні ***Draw graph*** повідомлення біля мітки «**Graph:»** зміниться на *“Contains parallel edges or self-loops”*. Аналогічне повідомлення буде виведено біля мітки **«MST:»** при натисканні кнопки **Draw MST**.

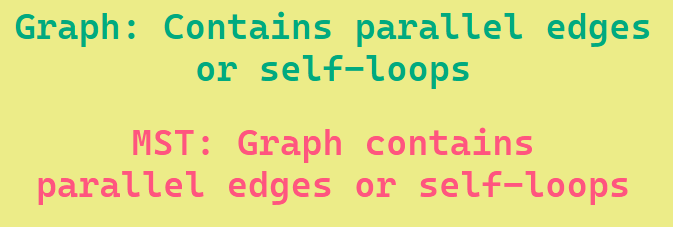


Рисунок 3.8 – Повідомлення про те, що завантажений граф містить кратні ребра або петлі

### 3.3 Опис тестових випадків та перевірка працездатності застосунку

Тестування працездатності застосунку здійснюватимемо за допомогою наступних тестових випадків:

1. Перевіримо роботу алгоритму для простого графа (таблиця 3.1).

Таблиця 3.1 – Тестовий випадок №1

|  |  |
| --- | --- |
| Вхідні дані | Очікуваний результат |
| 8  0 1 4 5 9  1 2 8 7 11  2 3 7 5 4  3 4 9 5 14  4 5 10  5 6 2 7 13  6 4 20 | 0 -- 1  1 -- 2  2 -- 5  5 -- 6  2 -- 3  3 -- 4  1 -- 7 |

Тест перевіряє, як працює застосунок та алгоритм Прима з простим графом, що не містить кратних ребер та петель. У даному випадку програма має успішно побудувати граф з вісьмома вершинами та дерево з ребрами, які наведено в таблиці 3.1.

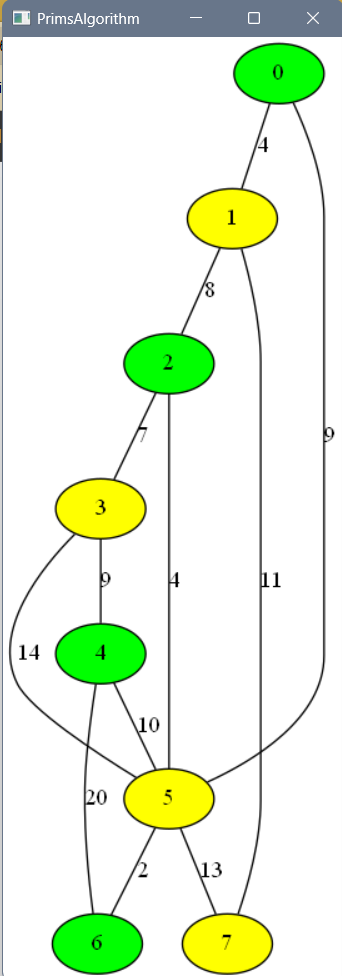


Рисунок 3.9 – Побудований граф для тесту №1

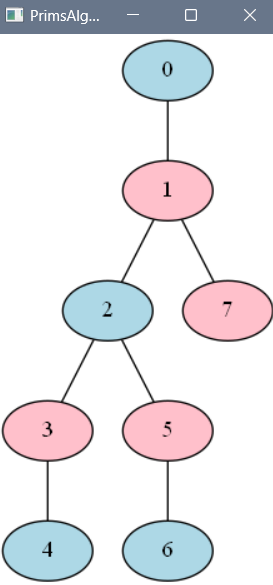


Рисунок 3.10 – Побудоване дерево для тесту №1

1. Перевіримо роботу алгоритму для іншого простого графа (таблиця 3.2).

Таблиця 3.2 – Тестовий випадок №2

|  |  |
| --- | --- |
| Вхідні дані | Очікуваний результат |
| 9  0 8 18 6 1 4 10 7 7  6 4 6 1 20  4 1 3 7 5  1 3 12  3 8 4 2 9  2 5 4 6 17  8 2 15 5 8 1 19 | 0 -- 6  6 -- 4  4 -- 1  4 -- 7  1 -- 3  3 -- 8  8 -- 5  5 -- 2 |

У даному випадку програма має успішно побудувати граф з дев’ятьма вершинами та дерево з ребрами, які наведено в таблиці 3.2.

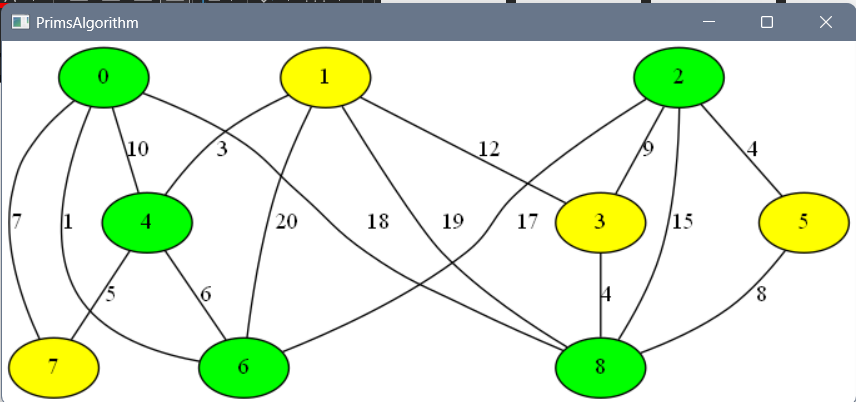


Рисунок 3.11 – Побудований граф для тесту №2

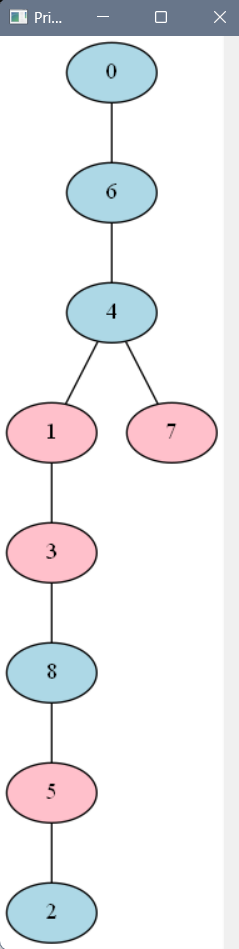


Рисунок 3.12 – Побудоване дерево для тесту №2

1. Перевіримо роботу алгоритму для графа, що містить кратні ребра (таблиця 3.3).

Таблиця 3.3 – Тестовий випадок №3

|  |  |
| --- | --- |
| Вхідні дані | Очікуваний результат |
| 7  0 1 2 3 3 6 4  1 2 3 4 2  3 4 5 4 7  4 6 6 5 7 | Повідомлення про те, що граф містить кратні ребра. |

Тест перевіряє, як працює застосунок та алгоритм Прима з графом, що містить кратні ребра. У даному випадку програма має вивести повідомлення про те, що завантажений граф містить кратні ребра, отже, не можливо знайти МКД.

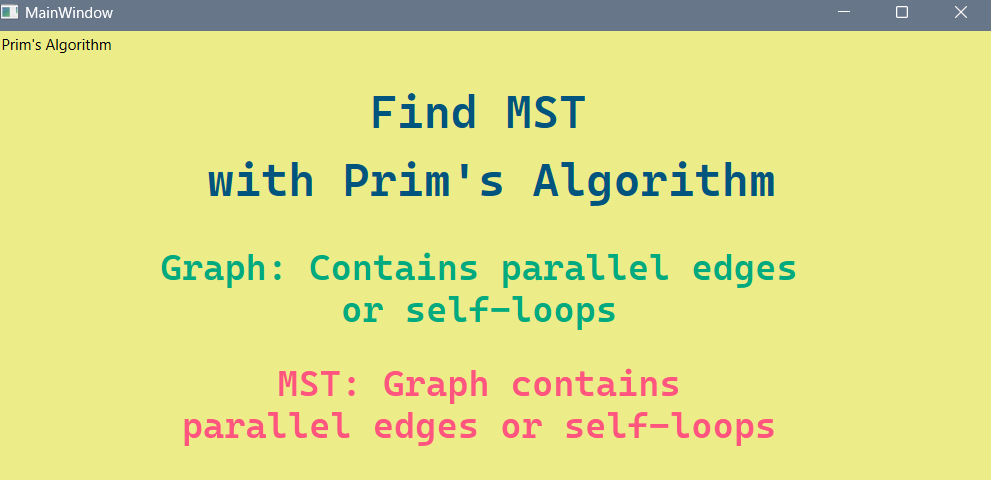


Рисунок 3.13 – Результат роботи застосунку для тесту №3

1. Перевіримо роботу алгоритму для графа, що містить петлі (таблиця 3.4).

Таблиця 3.4 – Тестовий випадок №4

|  |  |
| --- | --- |
| Вхідні дані | Очікуваний результат |
| 5  0 1 4 2 6  1 3 8 4 10  2 3 10 4 18 2 1  3 4 25  4 4 12 | Повідомлення про те, що граф містить петлі. |

Тест перевіряє, як працює застосунок та алгоритм Прима з графом, що містить петлі. У даному випадку програма має вивести повідомлення про те, що завантажений граф містить петлі, отже, не можливо знайти МКД.

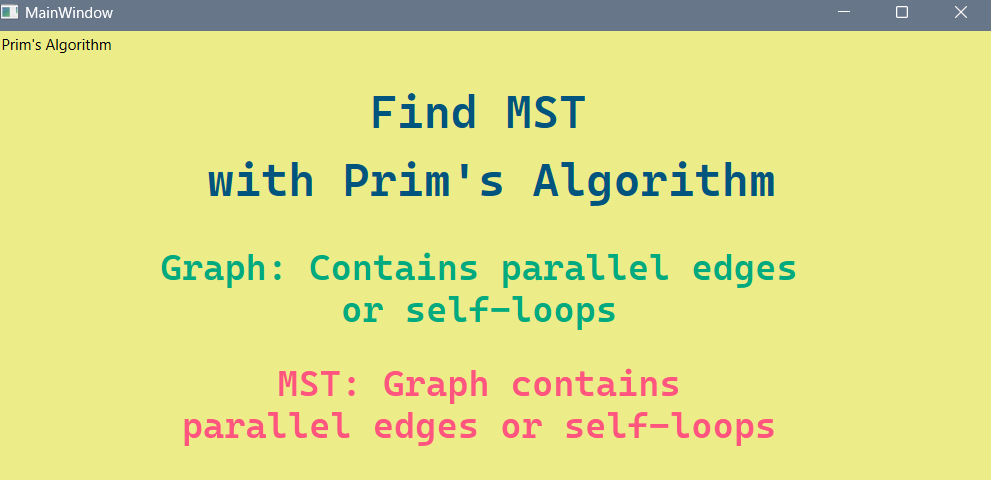


Рисунок 3.14 – Результат роботи застосунку для тесту №4

### Висновки до розділу 3

Реалізовано програмний продукт, що знаходить мінімальне кістякове дерево за алгоритмом Прима, відповідно до функціональних та нефункціональних вимог. Було сформовано тестові випадки, які перевірили роботу програмного застосунку для різних типів графів. Результати роботи програмного продукту підтверджують його працездатність. Застосунок правильно будує прості графи з різними кількостями вершин та знаходить МКД для них. У випадку, якщо користувач завантажує граф з паралельними ребрами чи петлями, для яких алгоритм Прима не працює, програмний продукт виводить попереджувальне повідомлення про те, що заданий граф містить кратні ребра чи петлі.

## ВИСНОВКИ

У ході виконання курсової роботи було проведено аналіз алгоритму Прима – жадібного алгоритму побудови мінімального кістякового дерева зваженого неорієнтованого зв’язного графа. Ознайомившись з теоретичною частиною алгоритму, передумовами для застосування, обмеженнями, перевагами та недоліками, було розроблено програмний продукт з графічним інтерфейсом користувача.

Програмний продукт було реалізовано відповідно до функціональних та нефункціональних вимог, визначених на етапі проєктування. Також були побудовані діаграми для спрощення та пришвидшення написання коду для програмного застосунку. Програмний продукт було протестовано, використовуючи у якості вхідних даних різні види графів, та підтверджено його працездатність.

Розроблений програмний продукт має зручний та інтуїтивний інтерфейс користувача з функціями завантаження файлу з графом, побудовою та збереженням графу, побудовою та збереженням мінімального кістякового дерева, знайденого за алгоритмом Прима. У детальному керівництві користувача було описано інструкції щодо користування програмним інтерфейсом, а також зазначено, з якими вхідними даними (видами графів) правильно працює програмний продукт.

Для подальшого вдосконалення можна розширити функціонал програмного інтерфейсу застосунку, оскільки даний варіант є досить простим та мінімалістичним. Також можна удосконалити функції побудови графа та мінімального кістякового дерева. Наприклад, забезпечити покрокову анімовану візуалізацію побудови вершин та ребер замість візуалізації на екрані вже повністю готового графа/МКД.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Introduction to Algorithms / T. H. Cormen et al. MIT Press, 2009. 1320 p.
2. Кормен, Т., Лейзерсон, Ч., Ріверст, Р., і Штайн, К. Алгоритми: побудова та аналіз, 3-є вид. - К.: Видавництво "Дія", 2013. - 1312 с. ISBN 978-617-708-074-5.
3. Sedgewick, R., & Wayne, K. Algorithms, 4th Edition. Addison-Wesley Professional, 2011.
4. Goodrich M. T., Tamassia R. Algorithm Design and Applications. Wiley & Sons, Incorporated, John, 2014.
5. DevDocs – C++ documentation [Електронний ресурс] : [Веб-сайт]. – DevDocs API Documentation. – Режим доступу: <https://devdocs.io/cpp/>.
6. Standard C++ [Електронний ресурс] : [Веб-сайт]. Standard C++. – Режим доступу: <https://isocpp.org/>.
7. DevDocs – Qt documentation [Електронний ресурс] : [Веб-сайт]. DevDocs API Documentation. – Режим доступу: <https://devdocs.io/qt/>.
8. Documentation [Електронний ресурс] : [Веб-сайт]. Graphviz. – Режим доступу: <https://graphviz.org/documentation/>.
9. Теорiя графiв. Лекцiї та варiанти iндивiдуальних домашнiх завдань [Текст]:

навчальний посiбник / I-26 С. П. Iглiн. – Харкiв: НТУ "ХПI" , 2017. – 146 с.

1. Теорія графів. [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра спеціальності 122 «Комп’ютерні науки»/ І.М. Кузьменко; КПІ ім. Ігоря Сікорського. — 2020. — 71 с.

## ДОДАТКИ

**Додаток А. Програмний код застосунку**

**graph.h**

#ifndef GRAPH\_H  
#define GRAPH\_H  
  
#include <sstream>  
#include <vector>  
#include <string>  
#include <utility>  
#include <unordered\_set>  
  
class Graph {  
public:  
 Graph();  
 ~Graph();  
 void readGraphFromFile(const std::string& filename);  
 bool hasParallelEdges() const;  
 void drawGraph(const std::string& outputFilename)const;  
 void clear();  
  
 std::vector<std::vector<std::pair<int, int>>> adjacencyList;  
  
private:  
 int vertexCount;  
};  
  
#endif *// GRAPH\_H*

**graph.cpp**

#include "graph.h"  
#include <fstream>  
#include <iostream>  
#include <graphviz/gvc.h>  
#include <queue>  
#include <unordered\_set>  
  
using namespace std;  
  
Graph::Graph() : vertexCount(0) {}  
  
Graph::~Graph() {}  
  
*// Reads a graph from a file*  
void Graph::readGraphFromFile(const std::string& filename) {  
 std::ifstream inFile(filename);  
 if (!inFile) {  
 std::cerr << "Unable to open the file: " << filename << std::endl;  
 return;  
 }  
  
 *// Read vertex count*  
 inFile >> vertexCount;  
 adjacencyList.resize(vertexCount);  
  
 *// Read edges*  
 std::string line;  
 std::getline(inFile, line); *// Read the rest of the line after vertexCount*  
 while (std::getline(inFile, line)) {  
 std::istringstream lineStream(line);  
 int vertex, neighbor, weight;  
 lineStream >> vertex;  
 while (lineStream >> neighbor >> weight) {  
 adjacencyList[vertex].push\_back(std::make\_pair(neighbor, weight));  
 adjacencyList[neighbor].push\_back(std::make\_pair(vertex, weight)); *// Add the opposite direction*  
 }  
 }  
  
 inFile.close();  
}  
  
bool Graph::hasParallelEdges() const {  
 for (int i = 0; i < vertexCount; i++) {  
 std::unordered\_set<int> neighbors;  
 for (const auto& neighbor : adjacencyList[i]) {  
 if (neighbor.first == i) {  
 return true; *// Found a self-loop*  
 }  
 if (neighbors.find(neighbor.first) != neighbors.end()) {  
 return true; *// Found a parallel edge*  
 }  
 neighbors.insert(neighbor.first);  
 }  
  
 for (const auto& neighbor : adjacencyList[i]) {  
 if (i < neighbor.first) {  
 continue; *// Already checked this direction*  
 }  
 std::unordered\_set<int> oppositeNeighbors;  
 for (const auto& oppositeNeighbor : adjacencyList[neighbor.first]) {  
 if (oppositeNeighbors.find(oppositeNeighbor.first) != oppositeNeighbors.end()) {  
 return true; *// Found a parallel edge in opposite direction*  
 }  
 oppositeNeighbors.insert(oppositeNeighbor.first);  
 }  
 }  
 }  
 return false;  
}  
  
*// Draws the graph using Graphviz library and saves it to a file*  
void Graph::drawGraph(const std::string& outputFilename) const {  
 GVC\_t \*gvc;  
 Agraph\_t \*graphViz;  
 Agedge\_t\* edge;  
  
 *// Initialize Graphviz context*  
 gvc = gvContext();  
 *// Create a new undirected graph*  
 graphViz = agopen(const\_cast<char\*>("Graph"), Agundirected, nullptr);  
  
 *// Add nodes to the graph*  
 std::vector<Agnode\_t\*> nodes(vertexCount);  
 for (int i = 0; i < vertexCount; i++) {  
 nodes[i] = agnode(graphViz, const\_cast<char\*>(std::to\_string(i).c\_str()), TRUE);  
 if (i % 2 == 0) {  
 agsafeset(nodes[i], const\_cast<char\*>("style"), const\_cast<char\*>("filled"), const\_cast<char\*>(""));  
 agsafeset(nodes[i], const\_cast<char\*>("fillcolor"), const\_cast<char\*>("green"), const\_cast<char\*>(""));  
 }  
 else {  
 agsafeset(nodes[i], const\_cast<char\*>("style"), const\_cast<char\*>("filled"), const\_cast<char\*>(""));  
 agsafeset(nodes[i], const\_cast<char\*>("fillcolor"), const\_cast<char\*>("yellow"), const\_cast<char\*>(""));  
 }  
 }  
  
 *// Add edges to the graph*  
 for (int i = 0; i < vertexCount; i++) {  
 bool directed = true;  
 for (const auto& neighbor : adjacencyList[i]) {  
 *// Check if the edge has already been drawn in the opposite direction*  
 if (i < neighbor.first || !directed) {  
 edge = agedge(graphViz, nodes[i], nodes[neighbor.first], nullptr, TRUE);  
 agsafeset(edge, const\_cast<char\*>("label"), const\_cast<char\*>(std::to\_string(neighbor.second).c\_str()), const\_cast<char\*>(""));  
 }  
 }  
 }  
  
 *// Apply layout and render the graph to a file*  
 gvLayout(gvc, graphViz, "dot");  
 gvRenderFilename(gvc, graphViz, "png", const\_cast<char\*>(outputFilename.c\_str()));  
 *// Cleanup*  
 gvFreeLayout(gvc, graphViz);  
 agclose(graphViz);  
 gvFreeContext(gvc);  
}  
  
void Graph::clear() {  
 vertexCount = 0;  
 adjacencyList.clear();  
}

**mst.h**

#ifndef MST\_H  
#define MST\_H  
  
#include "graph.h"  
#include <vector>  
#include <utility>  
  
class MST {  
public:  
 MST(const Graph &graph);  
 ~MST();  
  
 void getMST();  
 void drawMST(const std::string& outputFilename) const;  
 void saveToFile(const std::string& filename) const;  
  
private:  
 Graph graph;  
 std::vector<std::pair<int, int>> edges;  
};  
  
#endif *// MST\_H*

**mst.cpp**

#include "mst.h"  
#include <queue>  
#include <fstream>  
#include <iostream>  
#include <algorithm>  
#include <graphviz/gvc.h>  
  
using namespace std;  
  
*// Constructor initializes the MST object with the input graph*  
MST::MST(const Graph& g){  
 graph = g;  
}  
  
MST::~MST(){  
}  
  
void MST::getMST() {  
 const int n = graph.adjacencyList.size();  
 std::vector<bool> visited(n, false);  
 std::vector<int> parent(n, -1);  
 std::vector<int> key(n, std::numeric\_limits<int>::max());  
  
 *// Start from vertex 0*  
 key[0] = 0;  
 std::priority\_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::greater<std::pair<int, int>>> pq;  
 pq.push({0, 0});  
  
 while (!pq.empty()) {  
 auto [w, u] = pq.top();  
 pq.pop();  
 if (visited[u]) {  
 continue;  
 }  
 visited[u] = true;  
 if (parent[u] != -1) {  
 edges.push\_back({parent[u], u});  
 }  
 for (const auto& [v, weight] : graph.adjacencyList[u]) {  
 if (!visited[v] && weight < key[v]) {  
 key[v] = weight;  
 pq.push({key[v], v});  
 parent[v] = u;  
 }  
 }  
 }  
}  
  
*// Draws the Minimum Spanning Tree using Graphviz library and saves it to a file*  
void MST::drawMST(const std::string& outputFilename) const {  
 GVC\_t \*gvc;  
 Agraph\_t \*graphViz;  
  
 gvc = gvContext();  
 graphViz = agopen(const\_cast<char\*>("MST"), Agundirected, nullptr);  
  
 std::vector<Agnode\_t\*> nodes(graph.adjacencyList.size());  
 for (int i = 0; i < graph.adjacencyList.size(); i++) {  
 nodes[i] = agnode(graphViz, const\_cast<char\*>(std::to\_string(i).c\_str()), TRUE);  
 if (i % 2 == 0) {  
 agsafeset(nodes[i], const\_cast<char\*>("style"), const\_cast<char\*>("filled"), const\_cast<char\*>(""));  
 agsafeset(nodes[i], const\_cast<char\*>("fillcolor"), const\_cast<char\*>("#ADD8E6"), const\_cast<char\*>(""));  
 }  
 else {  
 agsafeset(nodes[i], const\_cast<char\*>("style"), const\_cast<char\*>("filled"), const\_cast<char\*>(""));  
 agsafeset(nodes[i], const\_cast<char\*>("fillcolor"), const\_cast<char\*>("pink"), const\_cast<char\*>(""));  
 }  
 }  
  
 for (const auto& edge : edges) {  
 agedge(graphViz, nodes[edge.first], nodes[edge.second], nullptr, TRUE);  
 }  
  
 gvLayout(gvc, graphViz, "dot");  
 gvRenderFilename(gvc, graphViz, "png", const\_cast<char\*>(outputFilename.c\_str()));  
 gvFreeLayout(gvc, graphViz);  
 agclose(graphViz);  
 gvFreeContext(gvc);  
}  
  
*// Saves the Minimum Spanning Tree edges to a file*  
void MST::saveToFile(const std::string& filename) const {  
 std::ofstream outFile(filename);  
 if (!outFile) {  
 std::cerr << "Unable to open the file: " << filename << std::endl;  
 return;  
 }  
  
 for (const auto& edge : edges) {  
 outFile << edge.first << " -- " << edge.second << std::endl;  
 }  
  
 outFile.close();  
}

**mainwindow.h**

#ifndef MAINWINDOW\_H  
#define MAINWINDOW\_H  
  
#include <iostream>  
#include <fstream>  
#include <vector>  
#include <queue>  
#include <utility>  
#include <QWidget>  
#include <QApplication>  
#include <QPushButton>  
#include <QVBoxLayout>  
#include <QFileDialog>  
#include <QLabel>  
#include <QMainWindow>  
#include <graphviz/gvc.h>  
  
#include "graph.h"  
#include "mst.h"  
  
QT\_BEGIN\_NAMESPACE  
namespace Ui { class MainWindow; }  
QT\_END\_NAMESPACE  
  
class MainWindow : public QMainWindow  
  
{  
 Q\_OBJECT  
  
public:  
 MainWindow(QWidget \*parent = nullptr);  
 ~MainWindow();  
  
private slots:  
 void openFile();  
 void visualizeGraph();  
 void saveGraph();  
 void visualizeMST();  
 void saveMST();  
  
private:  
 Ui::MainWindow \*ui;  
 Graph graph;  
 MST mst;  
};  
  
#endif *// MAINWINDOW\_H*

**mainwindow.cpp**

#include "mainwindow.h"  
#include "ui\_mainwindow.h"  
  
  
MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow), graph(), mst(graph) {  
 ui->setupUi(this);  
  
 *//Connect button signals to corresponding slots*  
 connect(ui->openButton, &QPushButton::clicked, this, &MainWindow::openFile);  
 connect(ui->drawGraphButton, &QPushButton::clicked, this, &MainWindow::visualizeGraph);  
 connect(ui->saveGraphButton, &QPushButton::clicked, this, &MainWindow::saveGraph);  
 connect(ui->drawMSTButton, &QPushButton::clicked, this, &MainWindow::visualizeMST);  
 connect(ui->saveMSTButton, &QPushButton::clicked, this, &MainWindow::saveMST);  
}  
  
MainWindow::~MainWindow()  
{  
 delete ui;  
}  
  
void MainWindow::openFile()  
{  
 graph.clear(); *//clear graph data before loading another one*  
 QString fileName = QFileDialog::getOpenFileName(this);  
 if (!fileName.isEmpty()) {  
 graph.readGraphFromFile(fileName.toStdString());  
 ui->graphLabel->setText("<center>Graph: Loaded</center>");  
 }  
}  
  
void MainWindow::visualizeGraph() {  
 if (graph.hasParallelEdges()) {  
 ui->graphLabel->setText("<center>Graph: Contains parallel edges<br>or self-loops</center>");  
 }  
  
 else {  
 *// Create a temporary file to save the graph image*  
 QString tempFilename = QDir::tempPath() + "/temp\_graph.png";  
 graph.drawGraph(tempFilename.toStdString());  
  
 *// Create a new image widget and set the image*  
 QPixmap pixmap(tempFilename);  
 QLabel\* imageLabel = new QLabel();  
 imageLabel->setPixmap(pixmap);  
  
 *// Create a new window to display the image*  
 QMainWindow\* graphWindow = new QMainWindow();  
 graphWindow->setCentralWidget(imageLabel);  
 graphWindow->show();  
  
 ui->graphLabel->setText("<center>Graph: Visualized</center>");  
 }  
}  
  
void MainWindow::saveGraph(){  
 QString outputFileName = "graph.png";  
 graph.drawGraph(outputFileName.toStdString());  
  
 ui->graphLabel->setText("<center>Graph: Saved as 'graph.png'</center>");  
}  
  
void MainWindow::visualizeMST() {  
 if (graph.hasParallelEdges()) {  
 ui->mstLabel->setText("<center>MST: Graph contains<br>parallel edges or self-loops</center>");  
 }  
  
 else {  
 *// Create a temporary file to save the mst image*  
 mst = MST(graph);  
 mst.getMST();  
 QString tempFilename = QDir::tempPath() + "/temp\_mst.png";  
 mst.drawMST(tempFilename.toStdString());  
  
 *// Create a new image widget and set the image*  
 QPixmap pixmap(tempFilename);  
 QLabel\* imageLabel = new QLabel();  
 imageLabel->setPixmap(pixmap);  
  
 *// Create a new window to display the image*  
 QMainWindow\* mstWindow = new QMainWindow();  
 mstWindow->setCentralWidget(imageLabel);  
 mstWindow->show();  
  
 ui->mstLabel->setText("<center>MST: Visualized</center>");  
 }  
}  
  
void MainWindow::saveMST() {  
 QString outputFilename = "mst.png";  
 mst.drawMST(outputFilename.toStdString());  
 mst.saveToFile("mst\_edges.txt");  
  
 ui->mstLabel->setText("<center>MST: Saved as 'mst.png' and<br>'mst\_edges.txt'</center>");  
}

**main.cpp**

#include "mainwindow.h"  
#include <QApplication>  
  
int main(int argc, char \*argv[])  
{  
 QApplication a(argc, argv);  
 MainWindow w;  
 w.show();  
 return a.exec();  
}

**mainwindow.ui**

<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0">  
 <class>MainWindow</class>  
 <widget class="QMainWindow" name="MainWindow">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>600</height>  
 </rect>  
 </property>  
 <property name="font">  
 <font>  
 <family>Cascadia Code SemiBold</family>  
 <pointsize>10</pointsize>  
 <bold>true</bold>  
 </font>  
 </property>  
 <property name="windowTitle">  
 <string>MainWindow</string>  
 </property>  
 <property name="autoFillBackground">  
 <bool>false</bool>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">background-color: rgb(236, 236, 136);  
border-color: rgb(0, 170, 127);  
border-top-color: rgb(0, 170, 127);</string>  
 </property>  
 <widget class="QWidget" name="centralwidget">  
 <widget class="QWidget" name="verticalLayoutWidget">  
 <property name="geometry">  
 <rect>  
 <x>130</x>  
 <y>310</y>  
 <width>526</width>  
 <height>201</height>  
 </rect>  
 </property>  
 <layout class="QVBoxLayout" name="verticalLayout">  
 <item>  
 <widget class="QPushButton" name="openButton">  
 <property name="styleSheet">  
 <string notr="true">font: 600 16pt &quot;Cascadia Code SemiBold&quot;;  
background-color: rgb(0, 170, 127);;  
color: rgb(255, 255, 255);</string>  
 </property>  
 <property name="text">  
 <string>Open graph file</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout\_2">  
 <item>  
 <widget class="QPushButton" name="drawGraphButton">  
 <property name="styleSheet">  
 <string notr="true">font: 600 16pt &quot;Cascadia Code SemiBold&quot;;  
color: rgb(255, 255, 255);  
background-color: rgb(255, 85, 127);  
</string>  
 </property>  
 <property name="text">  
 <string>Draw graph</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QPushButton" name="saveGraphButton">  
 <property name="styleSheet">  
 <string notr="true">font: 600 16pt &quot;Cascadia Code SemiBold&quot;;  
background-color: rgb(85, 170, 255);  
color: rgb(255, 255, 255);</string>  
 </property>  
 <property name="text">  
 <string>Save graph</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 <item>  
 <layout class="QHBoxLayout" name="horizontalLayout">  
 <item>  
 <widget class="QPushButton" name="drawMSTButton">  
 <property name="styleSheet">  
 <string notr="true">font: 600 16pt &quot;Cascadia Code SemiBold&quot;;  
background-color: rgb(85, 170, 255);  
color: rgb(255, 255, 255);</string>  
 </property>  
 <property name="text">  
 <string>Draw MST</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QPushButton" name="saveMSTButton">  
 <property name="styleSheet">  
 <string notr="true">font: 600 16pt &quot;Cascadia Code SemiBold&quot;;  
color: rgb(255, 255, 255);  
background-color: rgb(255, 85, 127);</string>  
 </property>  
 <property name="text">  
 <string>Save MST</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </item>  
 </layout>  
 </widget>  
 <widget class="QLabel" name="label">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>-40</y>  
 <width>801</width>  
 <height>201</height>  
 </rect>  
 </property>  
 <property name="font">  
 <font>  
 <family>Cascadia Code SemiBold</family>  
 <pointsize>28</pointsize>  
 <italic>false</italic>  
 <bold>true</bold>  
 </font>  
 </property>  
 <property name="styleSheet">  
 <string notr="true">font: 600 28pt &quot;Cascadia Code SemiBold&quot;;  
color: rgb(0, 85, 127);</string>  
 </property>  
 <property name="text">  
 <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p align=&quot;center&quot;&gt;Find MST &lt;/p&gt;&lt;p align=&quot;center&quot;&gt;with Prim's Algorithm&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>  
 </property>  
 </widget>  
 <widget class="QWidget" name="verticalLayoutWidget\_2">  
 <property name="geometry">  
 <rect>  
 <x>90</x>  
 <y>130</y>  
 <width>601</width>  
 <height>181</height>  
 </rect>  
 </property>  
 <layout class="QVBoxLayout" name="verticalLayout\_2">  
 <item>  
 <widget class="QLabel" name="graphLabel">  
 <property name="styleSheet">  
 <string notr="true">color: rgb(0, 170, 127);  
font: 600 22pt &quot;Cascadia Code SemiBold&quot;;  
</string>  
 </property>  
 <property name="text">  
 <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p align=&quot;center&quot;&gt;Graph: Not loaded&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>  
 </property>  
 </widget>  
 </item>  
 <item>  
 <widget class="QLabel" name="mstLabel">  
 <property name="styleSheet">  
 <string notr="true">color: rgb(255, 85, 127);  
font: 600 22pt &quot;Cascadia Code SemiBold&quot;;</string>  
 </property>  
 <property name="text">  
 <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p align=&quot;center&quot;&gt;MST: Not calculated&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>  
 </property>  
 </widget>  
 </item>  
 </layout>  
 </widget>  
 </widget>  
 <widget class="QMenuBar" name="menubar">  
 <property name="geometry">  
 <rect>  
 <x>0</x>  
 <y>0</y>  
 <width>800</width>  
 <height>25</height>  
 </rect>  
 </property>  
 <property name="font">  
 <font>  
 <family>Segoe UI</family>  
 <pointsize>9</pointsize>  
 <bold>false</bold>  
 </font>  
 </property>  
 <widget class="QMenu" name="menuPrim\_s\_Algorithm">  
 <property name="title">  
 <string>Prim's Algorithm</string>  
 </property>  
 </widget>  
 <addaction name="menuPrim\_s\_Algorithm"/>  
 </widget>  
 <widget class="QStatusBar" name="statusbar"/>  
 <widget class="QToolBar" name="toolBar">  
 <property name="windowTitle">  
 <string>toolBar</string>  
 </property>  
 <attribute name="toolBarArea">  
 <enum>TopToolBarArea</enum>  
 </attribute>  
 <attribute name="toolBarBreak">  
 <bool>false</bool>  
 </attribute>  
 </widget>  
 </widget>  
 <resources/>  
 <connections/>  
</ui>