НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра

автоматизованих систем обробки інформації та управління

**КУРСОВА РОБОТА**

|  |
| --- |
| з                                                           «Основ програмування»  (назва дисципліни)  на тему:    **«**ЗНАХОДЖЕННЯ НАЙКОРОТШОГО ОСТОВНОГО ДЕРЕВА» |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | Студента   І   курсу  ІП-51  групи Субцельного О. В.  напряму підготовки 6.050103 «Програмна інженерія»  спеціальності «Програмне забезпечення систем»  Керівник                   Головченко М. М.    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_асистент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                        (посада, вчене звання, науковий ступінь)    Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Кількість балів: \_\_\_\_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_\_ | | | Члени комісії | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (посада, вчене звання, науковий ступінь) | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (посада, вчене звання, науковий ступінь) | |  | | | |

Київ - 2016 рік

Національний технічний університет України “КПІ”

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації та управління

Дисципліна «Основи програмування»

Напрям "Програмна інженерія"

Курс 1 Група      ІП-51       Семестр 2

**ЗАВДАННЯ**

на курсову роботу студента

Субцельного Олександра Володимировича

(прізвище, ім’я, по батькові)

1. Тема роботи Знаходження найкоротшого остовного дерева

2. Строк здачі студентом закінченої роботи

3. Вихідні дані до роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

6. Дата видачі завдання

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника і студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи |  |  |
| 4. | Розробка сценарію роботи програми |  |  |
| 5. | Узгодження сценарію роботи програми з керівником |  |  |
| 6. | Розробка (вибір) алгоритму рішення задачі |  |  |
| 7. | Узгодження алгоритму з керівником |  |  |
| 8. | Узгодження з керівником інтерфейсу користувача |  |  |
| 9. | Розробка програмного забезпечення |  |  |
| 10. | Налагодження розрахункової частини програми |  |  |
| 11. | Розробка та налагодження інтерфейсної частини програми |  |  |
| 12. | Узгодження з керівником набору тестів для контрольного прикладу |  |  |
| 13. | Тестування програми |  |  |
| 14. | Підготовка пояснювальної записки |  |  |
| 15. | Здача курсової роботи |  |  |
| 16. | Захист курсової роботи |  |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Субцельний Олександр Володимирович

(підпис) (прізвище, ім’я, по батькові)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Головченко Максим Миколайович

(підпис) (прізвище, ім’я, по батькові)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

**АНОТАЦІЯ**

Пояснювальна записка до курсової роботи: 53 сторінки, 12 рисунків, 7 таблиць, 6 посилань.

**Об’єкт дослідження:** знаходження найкоротшого остовного дерева.

**Мета роботи:** розробка комплексу програм для пошуку найкоротшого остовного дерева методами Прима, Крускала, Борувки.

Дана курсова робота включає в себе: опис методів, застосування методу до конкретного завдання, код програми вирішення перерахованих вище методів на мові програмування С++, а також описання детального процесу розв’язання кожного з них.

Визначено порівняльну характеристику швидкості, ефективності та точності роботи методів. У висновках узагальнено підсумок проведеної роботи.

ЗНАХОДЖЕННЯ НАЙКОРОТШОГО ОСТОВНОГО ДЕРЕВА, МЕТОД ПРИМА, МЕТОД КРУСКАЛА, МЕТОД БОРУВКИ.

**ЗМІСТ**

ВСТУП ……………………………………………………………………………………6

1 ПОСТАНОВКА ЗАДАЧІ……………………………………………………………….7

2 ТЕОРЕТИЧНА ЧАСТИНА 8

2.1 Алгоритм Прима……...……………………………………………………………..8

2.2 Алгоритм Крускала…...…………………………………………………………….8

2.3 Алгоритм Борувки…………………………………………………………………..9

3 АЛГОРИТМИ РОЗВ’ЯЗАННЯ ЗАДАЧІ……………………………………………..10

3.1 Реалізація алгоритму………………………………………………………………10

3.2 Реалізація алгоритму методом Прима……..……………………………………..11

3.3 Реалізація алгоритму методом Крускала 12

3.4 Реалізація алгоритму методом Борувки 12

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 14

4.1 Функціональна схема програмного забезпечення 14

4.2 Опис функцій частин програмного забезпечення………………………………14

4.2.1 Функції користувача 15

4.2.2 Стандартні функції, використані в програмі……………………………...19

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 21

5.1 План тестування 21

5.2 Приклади тестування 21

5.2.1 Введений граф не є зв’язним……………………………………………….21

5.2.2 Користувач натискає кнопки, не ввівши попередньо граф....……………22

5.2.3 Повторне введення ребра……..…………………………………………….23

5.2.4 Перевірка правильності алгоритмів Прима, Крускала, Борувки………...23

6 ІНСТРУКЦІЯ КОРИСТУВАЧА 26

7 АНАЛІЗ І УЗАГАЛЬНЕННЯРЕЗУЛЬТАТІВ 31

7.1 Порівняння складності алгоритмів……………………………………………....32

7.2 Час виконання алгоритмів……………………………………………………...32

7.3 Кількість ітерацій алгоритмів……………………………………………………32

7.3 Перевірка правильності результату алгоритму Прима………………………...33

ВИСНОВКИ 35

ПЕРЕЛІК ПОСИЛАНЬ 36

ДОДАТКИ 37

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ………………………………………………...37

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ…………………………………………………..40

**ВСТУП**

В наш час нові інформаційні технології посідають дуже важливе місце не лише в спеціалізованих, але й в повсякденних сферах життя. Комп'ютери застосовуються в бізнесі, менеджменті, торгівлі, навчанні та багатьох інших сферах діяльності людини. Комп'ютерні технології дуже зручні для виконання різноманітних операцій, але в різних сферах застосування ці операції різні. Тому, кожна окрема галузь, яка використовує специфічні технічні засоби, потребує своїх власних програм, які забезпечують роботу комп'ютерів.

Розробкою програмного забезпечення займається така галузь науки, як програмування. Вона набуває все більшого й більшого значення останнім часом, адже з кожним днем комп'ютер стає все більш необхідним, все більш повсякденним явищем нашого життя.

При проектуванні залізниць, ліній електропередачі та інших ліній комунікації, інтернет мереж, трубопроводів виникає проблема побудови мережі з мінімальними витратами. В теорії графів таке завдання успішно вирішується шляхом побудови мінімального остовного дерева неорієнтованого графа. Дане завдання має кілька методів вирішення: метод Прима, метод Крускала, метод Борувки. У даній роботі представлена ​​програма, що базується на алгоритмах Прима, Крускала і Борувки, яка обчислює найкоротше остовне дерево неорієнтованого графа і виконує візуалізацію графа.

**1 ПОСТАНОВКА ЗАДАЧІ**

Розробити програмне забезпечення, що буде знаходити найкоротше остовне дерево, такими методами сортування як:

* Методом Прима;
* Методом Крускала;
* Методом Борувки.

**Вхідні дані:**

На вхід подається кількість вершин графу, кількість ребер, вага заданих ребер, які користувач вводить в таблицю діалогового вікна, або може зчитати з файлу graph.txt, та обирає метод вирішення задачі у діалоговому вікні.

**Вихідні дані:**

Таблиця, в якій присутні ребра остовного дерева, малюнок .png графу з усіма ребрами та ребрами остовного дерева, виділиними жирним контуром, файл з матрицею ребер остовного дерева, часом виконання програми, кількістю ітерацій.

**2 ТЕОРЕТИЧНА ЧАСТИНА**

**2.1 Алгоритм Прима**

Алгоритм Прима - алгоритм побудови мінімального остовного дерева зваженого зв'язного неорієнтованого графа. Алгоритм вперше був відкритий в 1930 році чеським математиком Войцехом Ярніка, пізніше перевідкритий Робертом Примом в 1957 році, і, незалежно від них, Е. Дейкстрой в 1959 році [1].

Алгоритм Прима володіє тією властивістю, що ребра в б множині G завжди утворюють єдине дерево. Дерево починається з довільної кореневої вершини і зростає до тих пір, поки не охопить всі вершини в графі. На кожному кроці до дерева G додається легке ребро, що з'єднує дерево G і окрему вершину з решти графа. Дане правило додає тільки безпечні для дерева G ребра; отже, по завершенні алгоритму ребра в дереві G утворюють мінімальне остовне дерево. Дана стратегія є жадібною, оскільки на кожному кроці до дерева додається ребро, яке вносить мінімально можливий внесок в загальну вагу.

Час роботи алгоритму Прима O(): для кожної ще не обраної будемо зберігати мінімальне ребро, провідне в уже обрану вершину [2].

Тоді, щоб на поточному кроці зробити вибір мінімального ребра, треба просто переглянути ці мінімальні ребра у кожній групі обраної ще вершини - асимптотика складе O (n).

Але тепер при додаванні в остов чергового ребра і вершини ці покажчики треба перераховувати. Зауважимо, що ці покажчики можуть тільки зменшуватися, тобто у кожній групі переглянутої ще вершини треба або залишити її покажчик без зміни, або привласнити йому вагу ребра в тільки що додану вершину. Отже, цю фазу можна зробити також за O (n).

Таким чином, ми отримали варіант алгоритму Прима з асимптотикою O().

**2.2 Алгоритм Крускала**

Алгоритм Крускала - алгоритм побудови мінімального остовного дерева зваженого зв'язного неорієнтованого графа. Алгоритм вперше описаний Джозефом Крускалом в 1956 році [3].

Алгоритм Крускала знаходить безпечне ребро для додавання в зростаючий ліс шляхом пошуку ребра з мінімальною вагою серед всіх ребер, що з'єднують два дерева в лісі. Алгоритм Крускала є жадібним, оскільки на кожному кроці він додає до лісу ребро з мінімально можливою вагою [4].

Час роботи алгоритму Крускала O(nm+m) : до початку роботи алгоритму необхідно впорядкувати ребра по вазі, це вимагає O (m) часу. Після чого компоненти зв'язності зручно зберігати в вигляді системи непересічних множин. Всі операції в такому випадку займуть O (m\*α(m,n)), де α - функція, обернена до функції Аккермана. Оскільки для будь-яких практичних завдань α(m,n) <5, то можна прийняти її за константу, таким чином, загальний час роботи алгоритму Крускала можна прийняти за O (m).

**2.3 Алгоритм Борувки**

Алгоритм Борувки - це алгоритм знаходження мінімального остовного дерева зваженого зв'язного неорієнтованого графа.

Вперше був опублікований в 1926 році Отакаром Борувка як метод знаходження оптимальної електричної мережі в Моравії. Кілька разів був перевідкритий, наприклад Флореком, Перкалом і Солліном [5].

Робота алгоритму складається з декількох ітерацій, кожна з яких складається в послідовному додаванні ребер до остовного лісу графа, до тих пір, поки ліс не перетвориться в дерево, тобто, ліс, що складається з однієї компоненти зв'язності. За кожний хід алгоритм додає до лісу ребра мінімальної ваги для кожного піддерева [6].

Час роботи алгоритму Борувки O(m+ ) : на кожній ітерації число дерев в остовному лісі зменшується принаймні в два рази, тому весь алгоритм працює не більше ніж за O() ітерацій. Кожна ітерація може бути реалізована зі складністю O(m), тому загальний час роботи алгоритму становить O(m) часу.

**3 АЛГОРИТМИ РОЗВ’ЯЗАННЯ ЗАДАЧІ**

**3.1 Реалізація алгоритму**

1. Початок
2. Відкрити діалогове вікно та заблокувати кнопки
3. Зчитати вершини графа та ваги ребер за допомогою спеціальних полів, де вводять опис графу та функції getEdges, розблокувати кнопки алгоритмів
4. Обрати алгоритм вирішення та розблокувати кнопки «Show» «Show Graph» «Write to output.txt»
   1. Якщо натиснути на кнопку «Prima»
      1. Якщо граф не зв’язний
         1. Вивести діалогове вікно з проханням ввести граф
      2. Інакше
         1. Запустити функцію Prima для реалізації алгоритму Прима
   2. Якщо натиснути на кнопку «Kruskala»
      1. Якщо граф не зв’язний
         1. Вивести діалогове вікно з проханням ввести граф
      2. Інакше
         1. Запустити функцію Kruskala для реалізації алгоритму Крускала
   3. Якщо натиснути на кнопку «Boruvka»
      1. Якщо граф не зв’язний
         1. Вивести діалогове вікно з проханням ввести граф
      2. Інакше
         1. Запустити функцію Boruvka для реалізації алгоритму Борувки
5. Якщо натиснути кнопку «Show»
   1. Вивести у таблицю остовне дерево графа, кількість ітерацій та час виконання обраного алгоритму
6. Якщо натиснути кнопку «Show Graph»
   1. Запустити функцію createFile та викликати додаток GraphViz для відмальовки графа.
7. Якщо натиснути кнопку «Write to output.txt»
   1. Запустити функцію WritetoOtput, щоб записати у текстовий файл output.txt остовне дерево графа
8. Якщо натиснути кнопку «Delete graph»
   1. Знищити опис графу та очистити таблиці
9. Якщо натиснути кнопку «Close»
   1. Закрити діалогове вікно
10. Кінець

**3.2 Реалізація алгоритму методом Прима**

1. Початок
2. Створюємо масив з «кольорами вершин» на перевірку відвідуваних вершин, масив мінімальних відстаней, масив остовного дерева
3. Створення матриці з вагою кожного ребра
4. Створюємо змінні start, min\_edge,from,to
5. Проходимо по всім вершинам:
   1. min\_edge присвоїти INF
   2. Проходимо по всім вершинам [цикл по j]
      1. Якщо ребро з вершини start в j менше за мінімальну відстань для вершини j
         1. Мінімальну відстань для j присвоїти ребру з start в j
      2. Якщо min\_edge більше за мінімальну відстань для j і вершина j не відвідана
         1. min\_edge присвоїти мінімальній відстані для j
         2. Змінній from присвоїти вершину з якої виходить ребро мінімальної ваги
         3. Змінній to присвоїти змінну j
   3. Змінній start присвоїти змінну to
   4. Позначити вершину to як пройдену в масиві «кольорів»
   5. Додати до масиву остовного дерева вершини from та to
6. Кінець

**3.3 Реалізація алгоритму методом Крускала**

1. Початок
2. Створення та визначення масиву з полями для ваги ребра та вершин цього ребра table
3. Створення масиву остовного дерева
4. Сортування масиву table за значеннями ваги ребер
5. Створення масиву масиву tree\_id для збереження номера дерева, якому належить вершина та заповнення значення номером кожної вершини
6. Проходження по всім відсортованим ребрам графа [цикл по m]
   1. Присвоїти змінній a номер вершини виходу
   2. Присвоїти змінній b номер вершини заходу
   3. Якщо дві вершини не належать одному дереву
      1. Додати до масиву остовного дерева вершини a i b
      2. Присвоїти змінній old\_id номер дерева, в якій знаходиться b
      3. Присвоїти змінній new\_id номер дерева, в якій знаходиться a
      4. Проходження по масиву tree\_id [цикл по n]
         1. Присвоєння елементу дерева з значенням old\_id значення new\_id
7. Кінець.

**3.4 Реалізація алгоритму методом Борувки**

1. Початок
2. Створення та визначення масиву з полями для ваги ребра та вершин цього ребра table
3. Створення масиву остовного дерева, лічильника ccount, масиву a
4. Сортування масиву table за значеннями ваги ребер
5. Створення масиву tree\_id для збереження номера дерева, якому належить вершина та заповнення значення номером кожної вершини
6. Поки лічильник ccount не дорівнює n-1( де n – кількість вершин графа)
   1. Створення змінних k=0(лічильник кількості пройдених вершин) ,l=-1( лічильник проходу по елементам відсортованого масиву table), та масиву «кольорів вершин» на перевірку відвідуваності
   2. Поки k не дорівнює n-ccount
      1. Інкрементуємо змінну l
      2. Якщо вершина, що відповідає лічильнику l не відвідувана та номери дерев, яким відповідають вершини, що відповідають лічильнику l не рівні
         1. Помічаємо вершину як помічену
         2. Інкрементуєм k
         3. Ставимо у відповідність елементу масива a з індексом першої вершини значення другої вершини одного ребра
   3. Заповнення масиву остовного дерева [цикл по n]
      1. Якщо номер дерева лічильника i не дорівнює номеру дерева значення масиву a індексу лічильника i
         1. Додати до масиву остовного дерева вершини з номерами лічильника i та значенню масиву a з індексом лічильника i
      2. Присвоїти змінній old\_id номер лічильника i
      3. Присвоїти змінній new\_id номер дерева значення масиву a з індексом лічильника i
      4. Проходження по масиву tree\_id [цикл по n]
         1. Присвоєння елементу дерева з значенням old\_id значення new\_id
      5. Інкрементувати ccount
7. Кінець.

**4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**4.1 Функціональна схема програмного забезпечення**

Програмний додаток містить головний файл, 5 файлів реалізації, 6 заголовних файлів та заголовний файл для реалізації інтерфейсу (Рисунок 4.1).

graph.cpp

graph.h

WriteRead.h

variables.h

boruvka.h

prima.h

kruskala.h

researchGraph.h

kruskala.cpp

prima.cpp

boruvka.cpp

WriteRead.cpp

researchGraph.cpp

Рисунок 4.1 – Функціональна схема програмного забезпечення

**4.2 Опис функцій частин програмного забезпечення**

Програма поділена на наступні модулі:

* *kruskala.h* – реалізує алгоритм Крускала ;
* *prima.h* – реалізує алгоритм Прима;
* *boruvka.h* – реалізує алгоритм Борувки;
* *WriteRead.h* – реалізує зчитування з файлу, запис у файл результату, створює .dot файл для візуалізації графу;
* *variables.h* – описані основні змінні.
* *researchGraph.h – реалізовує перевірку на зв’язність графу*
* *Graph.h – реалізує інтерфейс користувача*

**4.2.1 Функції користувача**

Функції користувача описані у таблиці 4.1

Таблиця 4.1 - Функції користувача

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1. | *Prima* | Знаходження мінімального остовного дерева методом Прима | Кількість вершин, ребер,масив опису графу, змінні,що зберігають час роботи та кількість ітерацій | Масив певного розміру з описом остовного дерева | *Prima.h* |
| 2. | *kruskala* | Знаходження мінімального остовного дерева методом Крускала | Кількість вершин, ребер,масив опису графу, змінні,що зберігають час роботи та кількість ітерацій | Масив певного розміру з описом остовного дерева | *kruskala.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 3. | *boruvka* | Знаходження мінімального остовного дерева методом Борувки | Кількість вершин, ребер, масив опису графу, змінні,що зберігають час роботи та кількість ітерацій | Масив певного розміру з описом остовного дерева | *boruvka.h* |
| 4. | *research* | Перевірка на зв’язність графу | Кількість вершин, ребер, масив опису графу, масив з додатковими номерами вершин графу | True/false в залежності від того, чи є граф зв’язним | *researchGraph.h* |
| 5. | *createFile* | Створення .dot файлу для подальшої візуалізації графу | Кількість вершин, ребер, масив опису графу, масив з остовним деревом | Файл .dot | *WriteRead.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 6. | *writetoOutput* | Запис остовного дерева, часу роботи алгоритму і  кількість ітерацій у файл output.txt | Кількість вершин, ребер,масив остовного дерева, змінні,що зберігають час роботи та кількість ітерацій | Файл output.txt | *WriteRead.h* |
| 7. | *getEdges* | Знаходження кількості вершин і ребер, та створення масиву з додатковими номерами для вершин графу | - | - | *graph.h* |
| 8. | *on\_okButton\_clicked* | Обробка події натискання кнопки «Enter» | - | - | *graph.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 9. | *on\_boruvkaButton\_clicked* | Обробка події натискання кнопки «Boruvka» | - | - | *graph.h* |
| 10. | *on\_kruskalaButton\_clicked* | Обробка події натискання кнопки «Kruskala» | - | - | *graph.h* |
| 11. | *on\_primaButton\_clicked* | Обробка події натискання кнопки «Prima» | - | - | *graph.h* |
| 12. | *on\_pushButton\_3\_clicked* | Обробка події натискання кнопки «Show» | - | - | *graph.h* |
| 13. | *on\_pushButton\_4\_clicked* | Обробка події натискання кнопки «Show Graph» | - | - | *graph.h* |
| 14. | *on\_pushButton\_clicked* | Обробка події натискання кнопки «Delete» | - | - | *graph.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 15. | *on\_pushButton\_6\_clicked* | Обробка події натискання кнопки «Write to output.txt» | - | - | *graph.h* |

**4.2.2 Стандартні функції, використані в програмі**

Стандартні функції, використані в програмі, описані у таблиці 4.2

Таблиця 4.2 - Стандартні функції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1. | *Push\_back* | Додати в кінець масиву структури «вектор» елемент переданий у дужках | Елемент типу відповідного до типу вектора | - | *vector* |
| 2. | *sort* | Сортує елементи в діапазоні [перший, останній) в порядку зростання | Початковий та кінцевий елементи для сортування | - | *algorithm* |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 3. | *make\_pair* | Створює об’єкт пари з першим і другим елементами | Значення членів першої і другої, відповідно, об’єкта пари | Об’єкт пара, елементи якої перший і другий елементи | *utility* |
| 4. | *clock* | Повертає кількість тимчасових тактів, що минули з початку запуску програми | - | К-сть тактів що минули з моменту запуску програми | *time.h* |

**5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**5.1 План тестування**

Усі можливі випадки виникнення помилок у програмі залежать від вхідних даних, тобто тої інформації, що отримується від користувача. Тому тестування програми полягає у виявленні правильності та коректності обробки програмою різних вхідних даних.

Задля виявлення усіх похибок роботи програми потрібно запустити виконання алгоритмів в таких умовах:

* Граф, що введений, не є зв’язним
* Користувач натисне на кнопки не в правильному порядку
* Повторне введення ребра
* Перевірка правильності роботи алгоритмів Прима, Крускала, Борувки.

Для вирішення відповідних помилкових ситуацій слід створити додаткові функції для перевірки на правильність введення графу і блокування кнопок.

Далі слід упевнитись, що усі методи програми коректно працюють на усіх наборах вхідних значень.

**5.2 Приклади тестування**

**5.2.1 Граф, введений користувачем, не є зв’язним**

Якщо користувач введе граф, який не є зв’язним, під час натиску кнопки з назвою алгоритму на екрані з’явиться вікно з помилкою та проханням ввести зв’язний граф (Рисунок 5.1).

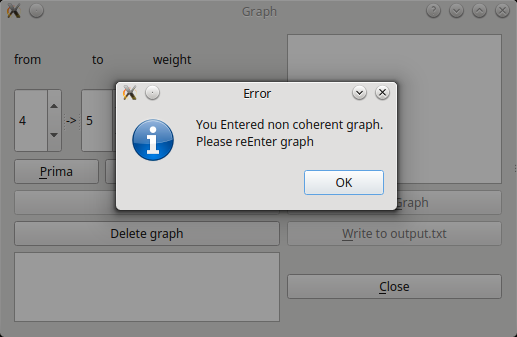


Рисунок 5.1 – Вікно з проханням ввести зв’язний граф

**5.2.2 Користувач натисне на кнопки, не вводивши попередньо граф**

Якщо користувач натисне на кнопку рішення, або виведення малюнку графу не ввівши попередньо граф виникне помилка. Задля цього кнопки блокуються, доки програма не отримає опис графу (Рисунок 5.2).

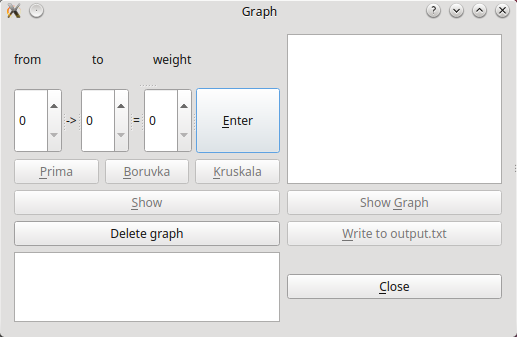


Рисунок 5.2 – Блокування кнопок

Надалі розблокується кнопка алгоритмів. Після того, як програма опрацює граф, можна натиснути на кнопки, що виведе остовне дерево у таблицю, відмалює граф, та записує результат у файл.

**5.2.3 Повторне введення ребра**

При повторному введені ребра, ребро перезаписується, із зміною ваги на нову, введену користувачем. Помилки не відбувається. Програма не зупиняє свою роботу.

**5.2.4 Перевірка правильності роботи алгоритмів Прима, Крускала, Борувки**

На вхід подається опис графу(вихідна вершина, вершина в яку входить ребро та вага ребра). За натискання кнопки «Prima», «Kruskala», «Boruvka» програма має опрацювати граф та записати у масив MST остовне дерево графу. За натисканням кнопки «Show» програма має вивести остовне дерево у таблицю діалогового вікна. За натисканням кнопки «Show Graph» програма має вивести малюнок графу з ребрами остовного дерева, що наведені жирними лініями. Тести описані у таблиці 5.1, таблиці 5.2, таблиці 5.3.

Таблиця 5.1 - Перевірка роботи алгоритму Прима

|  |  |
| --- | --- |
| Мета тесту | Перевірити алгоритм Прима при введені коректних даних |
| Початковий стан системи | Відкрите вікно програми |
| Вхідні дані | Опис графа |
| Схема проведення тесту | Введення даних та натискання кнопок «Prima» «Show» |
| Очікуваний результат | Таблиця з остовним деревом, часом роботи алгоритму і кількістю ітерацій |
| Стан системи після проведення випробувань | Виведено таблицю з остовним деревом графа, кількість ітерацій та час роботи алгоритму |

Таблиця 5.1 - Перевірка роботи алгоритму Крускала

|  |  |
| --- | --- |
| Мета тесту | Перевірити алгоритм Крускала при введені коректних даних |
| Початковий стан системи | Відкрите вікно програми |
| Вхідні дані | Опис графа |
| Схема проведення тесту | Введення даних та натискання кнопок «Kruskala» «Show» |
| Очікуваний результат | Таблиця з остовним деревом, часом роботи алгоритму і кількістю ітерацій |
| Стан системи після проведення випробувань | Виведено таблицю з остовним деревом графа, кількість ітерацій та час роботи алгоритму |

Таблиця 5.1 - Перевірка роботи алгоритму Борувки

|  |  |
| --- | --- |
| Мета тесту | Перевірити алгоритм Борувки при введені коректних даних |
| Початковий стан системи | Відкрите вікно програми |
| Вхідні дані | Опис графа |
| Схема проведення тесту | Введення даних та натискання кнопок «Boruvka» «Show» |
| Очікуваний результат | Таблиця з остовним деревом, часом роботи алгоритму і кількістю ітерацій |
| Стан системи після проведення випробувань | Виведено таблицю з остовним деревом графа, кількість ітерацій та час роботи алгоритму |

Отриманий результат є очікуваним. При правильно введених даних, програма не видає помилки, та не припиняє роботу. Усі три алгоритми вивели однаковий результат на однаковому тестовому графі(Рисунок 5.3). Отже алгоритми працюють правильно.

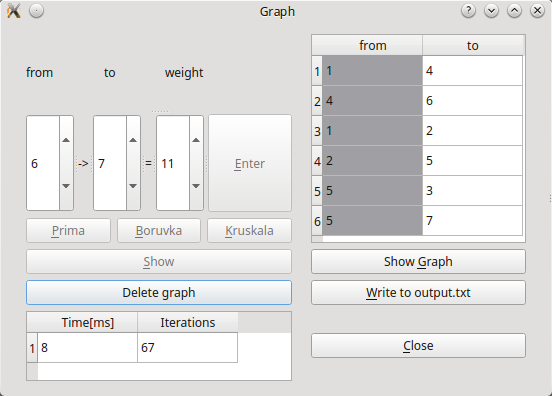


Рисунок 5.3 – Результат роботи алгоритму

**6 ІНСТРУКЦІЯ КОРИСТУВАЧА**

Після запуску виконавчого файлу (подвійний клік по іконці файлу в операційній системі) з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

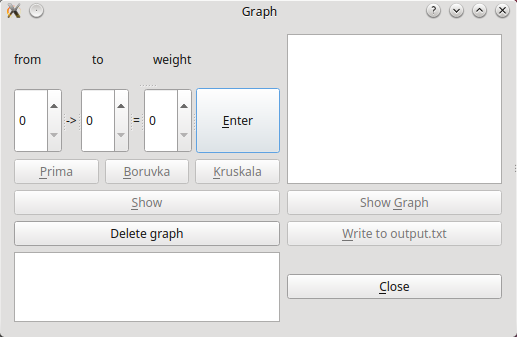


Рисунок 6.1 – Головне вікно програми

Далі знизу є кнопка «Close» , щоб закрити програму. Зверху «Close» присутні поля для введення ребер графу.

Після введення опису графу розблокуються кнопки «Prima» «Kruskala» «Boruvka». Слід обрати одну з клавіш, для того, щоб програма знайшла мінімальне остовне дерево обраним методом. (Рисунок 6.2).

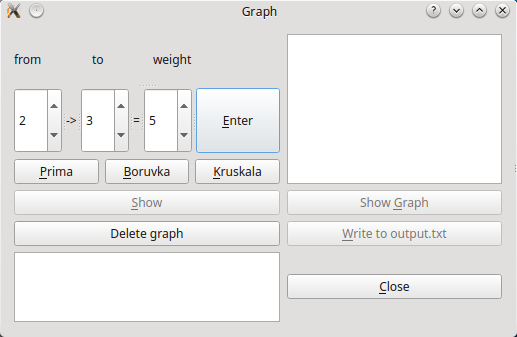


Рисунок 6.2 – Розблоковані кнопки алгоритмів

При введені зв’язного графу, надалі розблокуються наступні кнопки «Show», що відповідає за запис у таблицю діалогового вікна, що знаходиться справа остовного дерева та блокує кнопки «Enter» «Prima» «Kruskala» «Boruvka» «Show» (Рисунок 6.3).

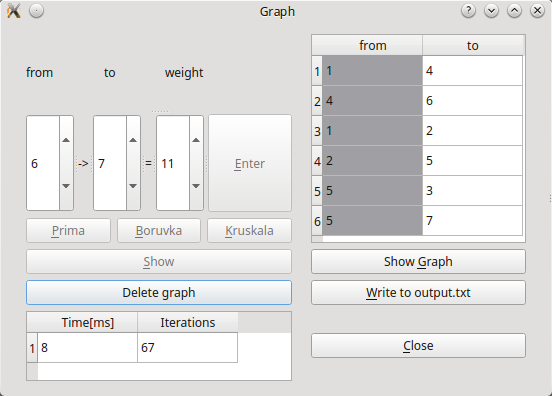


Рисунок 6.3 – Результат роботи програми

«Show Graph» - відображає малюнок графу з ребрами та жирним наведеним остовним деревом (Рисунок 6.4).

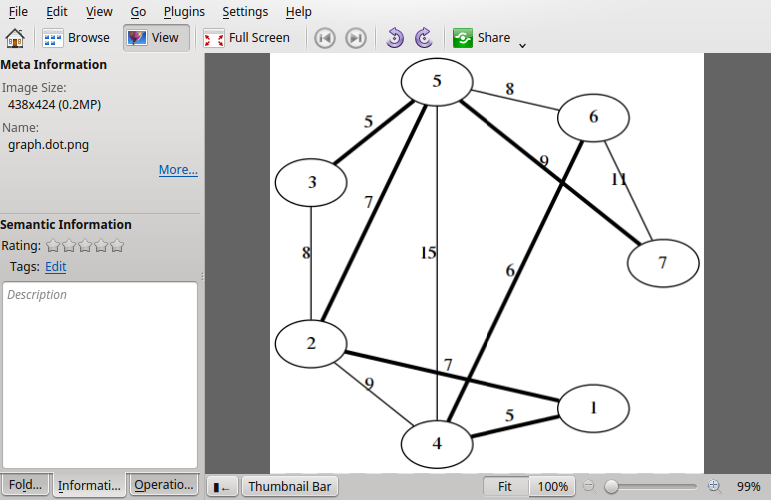


Рисунок 6.4 – Малюнок графу

«Write to output.txt» - запис остовного дерева, кількості ітерацій та часу роботи алгоритму у текстовий файл output.txt. (Рисунок 6.5)

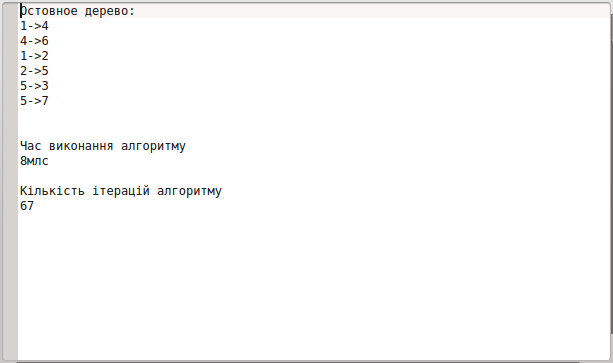


Рисунок 6.5 – Вихідний файл

У програмі присутні гарячі клавіші. При комбінації Alt+перша літера кнопки натискається відповідна кнопка.

Якщо користувач ввів не зв’язний граф, з’явится діалогове вікно, що повідомить користувача, про невірність введення даних. (Рисунок 6.6).

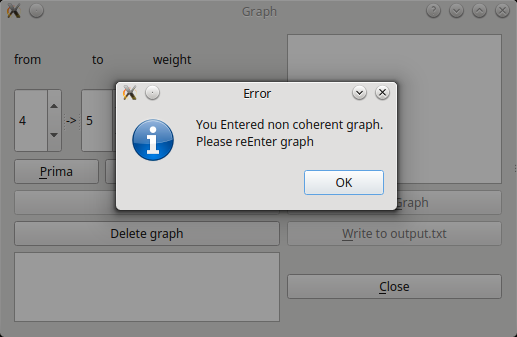


Рисунок 6.6 – Вікно з проханням ввести зв’язний граф

Всі результати, що були виведені у файл під час виконання програми та малюнок графу залишаться незмінними.

**7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ**

**7.1 Порівняння складності алгоритмів**

Складність алгоритму Прима співпадає з теоретичною складністю алгоритму.

Алгоритм Крускала реалізований в даній роботі має складність O(nm+m): cортування ребер потребує O(m) операцій. Приналежність вершини того чи іншого піддерева зберігається просто за допомогою масиву tree\_id - в ньому для кожної вершини зберігається номер дерева, якому вона належить. Для кожного ребра ми за O (1) визначаємо, чи належать його кінці різним деревам. Нарешті, об'єднання двох дерев здійснюється за O (n) операцій простим проходом по масиву tree\_id. З огляду на що всього операцій об'єднання m, ми і отримуємо асимптотики O(nm+m).

Складність алгоритму вища за теоретичну, що пояснюється додатковими операціями, такими як сортування масиву, та зберігання id кожної вершини в масиві tree\_id.

Алгоритм Борувки, реалізований в даній роботі, має складність O(m+): cортування ребер потребує O (m) операцій. На кожній ітерації число дерев в остовному лісі зменшується принаймні в два рази, тому весь алгоритм робить не більше O () ітерацій. Приналежність вершини того чи іншого піддерева зберігається просто за допомогою масиву tree\_id - в ньому для кожної вершини зберігається номер дерева, якому вона належить. Кожна ітерація може бути реалізована зі складністю O (n), об'єднання двох дерев здійснюється за O (n) операцій простим проходом по масиву tree\_id, тому загальна складність роботи алгоритми становить O(m+ ).

Складність алгоритму вища за теоретичну, що пояснюється додатковими операціями, такими як сортування масиву, зберігання id кожної вершини в масиві tree\_id.

**7.2 Час виконання алгоритмів**

Час виконання алгоритмів наведений у таблиці 7.1

Таблиця 7.1 - Час виконання алгоритмів

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Граф 5 вершин | Граф 10 вершин | Граф 15 вершин |
| Прима | 3 мс | 5 мс | 7 мс |
| Крускала | 4 мс | 7 мс | 11 мс |
| Борувки | 6 мс | 10 мс | 13 мс |

Залежність часу роботи алгоритмів від кількості вершин наведений на графіку (Рисунок 7.1).

Рисунок 7.1 - Залежність часу роботи алгоритму від кількості вершин

**7.3 Кількість ітерацій алгоритмів**

Кількість ітерацій алгоритмів наведена у таблиці 7.2

Таблиця 7.2 - Кількість ітерацій алгоритмів

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Граф 5 вершин | Граф 10 вершин | Граф 15 вершин |
| Прима | 35 | 119 | 254 |
| Крускала | 41 | 141 | 296 |
| Борувки | 46 | 152 | 312 |

Залежність часу роботи алгоритмів від кількості вершин наведений на графіку (Рисунок 7.2).

Рисунок 7.2 - Залежність кількості ітерацій алгоритмів від кількості вершин

**7.4 Перевірка правильності результату алгоритму Прима**

Перевіримо правильність результатів алгоритму за допомогою інтернет ресурсів.

Використаємо інтернет-ресурс Вікіпедія, на якому є знайдене остовне дерево графу. Введемо аналогічний граф у програму.

На вхід подається граф з наступними параметрами: вершина, з якої виходить ребро, в яку входить ребро та вага ребра:

[1 2 7],[1 4 5],[2 3 8],[2 5 7],[2 4 9],[3 5 5],[4 5 15],[4 6 6],[5 6 8],[5 7 9],[6 7 11]

Результатом є остовне дерево наступного вигляду:

[1 2],[1 4],[2 5],[3 5],[4 6],[5 7]

В результаті роботи програми методом алгоритму Прима отримаємо наступний результат:

[1 4],[4 6],[1 2],[2 5],[5 3],[5 7]

Час роботи алгоритму: 6мс. Кiлькiсть iтерацiй: 67.

В результаті роботи програми методом алгоритму Крускала отримаємо наступний результат:

[1 4],[3 5],[4 6],[1 2],[2 5],[5 7]

Час роботи алгоритму: 8мс. Кiлькiсть iтерацiй: 83.

В результаті роботи програми методом алгоритму Борувки отримаємо наступний результат:

[4 1],[1 2],[5 3],[4 6],[5 7],[5 2]

Час роботи алгоритму: 10мс. Кiлькiсть iтерацiй: 102.

Усі три алгоритми виводять аналогічний результат ідентичний результату з інтернет-ресурсу Вікіпедія.  
 Алгоритми працюють правильно.

**ВИСНОВКИ**

На етапі ознайомлення з теоретичними аспектами завдання даної курсової роботи було досліджено алгоритми знаходження остовного дерева, проаналізовано їх складність, достовірність та доцільність використання в межах реальних проектів, що було доведено реалізацією програмного забезпечення, яке розглядається в данні курсовій роботі.

На етапі проектування програмного забезпечення реалізовано алгоритми Прима, Крускала, Борувки. В результаті виконання курсової роботи були вдосконалені навички програмування на мові С++. Вивчено багатоплатформовий інструментарій розробки ПЗ на мові С++ Qt.

Окрім реалізації саме алгоритму задачі, було розроблено користувацький інтерфейс, що полегшує взаємодію користувача з програмою та значно розширює цільову аудиторію використання програми. Також була розроблена інструкція користувача, що дає можливість будь-кому зрозуміти основні етапи роботи з програмою та вже за декілька хвилин ознайомлення використовувати продукт для поставлених цілей.

Було проведене тестування створеної програми, під час якого було визначено, що програма правильно обробляє вхідні дані, опрацьовує дані, виводить результати та успішно опрацьовує окремі ситуації, при некоректному введені данних. Також було виконано аналіз отриманих результатів та підтверджено, що алгоритми працюють коректно і очікувано.

Результатом курсової роботи є створенний програмний проект, який можна використовувати у навчальних цілях, може використовуватись для мінімізації витрат при прокладанні інтернет мереж, трубопроводів, ліній електропередачі і комунікації, залізниць

**ПЕРЕЛІК ПОСИЛАНЬ**

1. Библиотека алгоритмов на графах. Теория – алгоритмы – примеры – программы: Алгоритм Прима [Електронний ресурс] - Режим доступу: http://urban-sanjoo.narod.ru/prim.html.

2. MAXimal (Алгоритмы, олимпиадное программирование, математика): Минимальное остовное дерево. Алгоритм Прима [Електронний ресурс] / під ліцензією Public Domain – 2008. – Режим доступу: http://e-maxx.ru/algo/mst\_prim.

3. MAXimal (Алгоритмы, олимпиадное программирование, математика): Минимальное остовное дерево. Алгоритм Крускала [Електронний ресурс] / під ліцензією Public Domain – 2008. – Режим доступу: http://e-maxx.ru/algo/mst\_kruskal.

4. Библиотека алгоритмов на графах. Теория – алгоритмы – примеры – программы: Алгоритм Крускала [Електронний ресурс] – Режим доступу:<http://urban-sanjoo.narod.ru/kruskal.html>.

5. Википедия свободная энциклопедия: Алгоритм Борувки [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Алгоритм\_Борувки.

6. Построение минимального остовного дерева алгоритмом Борувки. Программная реализация [Електронний ресурс] / Гаврилов А. И., Тишин В. В. // Вестник науки и образования – 2015. - № 4. – Режим доступу до журн.: http://cyberleninka.ru/article/n/postroenie-minimalnogo-ostovnogo-dereva-algoritmom-boruvki-programmnaya-realizatsiya.

Додаток А Технічне завдання

МІНІСТЕРСТВО НАУКИ І ОСВІТИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

Керівник Головченко Максим Миколайович

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

Виконавець:

Студент Субцельний Олександр Володимирович

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Знаходження найкоротшого остовного дерева»

з дисципліни:

«Основи програмування»

Київ 2016

1.1 Мета: Метою курсової роботи є розробка комплексу програм для пошуку найкоротшого остовного дерева методами Прима, Крускала, Борувки.

1.2 Найменування та галузь застосування об'єкта розробки:Дана робота присвячена розробці програмного забезпечення для знаходження найкоротшого остовного дерева в графі в галузі комунікації.

1.3 Підстава для проведення робіт:Підставою для розробки програмного забезпечення є навчальний план спеціальності 6.050103 «Програмна інженерія», робоча програма дисципліни „Основи програмування”, індивідуальне завдання.

1.4 Дата початку роботи: «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2016 р.

1.5 Дата закінчення роботи: «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2016 р.

1.6 Призначення розробки: Функціональним призначенням програми є автоматизація наступних процесів: знаходження найкоротшого остовного дерева графу, графічна інтерпретація графу та остовного дерева, збереження результатів у файл та аналіз ефективності методів. Програма може експлуатуватися для мінімізації витрат при прокладанні інтернет мереж, трубопроводів, ліній електропередачі і комунікації, залізниць.

1.7 Вимоги до програми та програмної документації: Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ИСО 5807 - 85 ГОСТ на розробку програмних документів, схем алгоритмів програм, даних та систем.

ГОСТ 19.781 - 74 - Вимоги до розробки програмного забезпечення.

ГОСТ 19.101-77 (СТ СЭВ 1626 - 79) - Держстандарт на розробку програмної документації, видів програм та програмних документів.

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 95 - Розробка технічної документації.

<*інші вимоги до функціональних характеристик, надійності, складу і параметрів технічних засобів тощо*>

1.8 Стадії та етапи розробки:

1. Аналіз методів вирішення поставленої задачі (до \_\_.\_\_.2016 р.)

2. Розробка сценарію роботи програми (до \_\_.\_\_.2016 р.)

3. Розробка алгоритмічного забезпечення (до \_\_.\_\_.2016 р.)

4. Розробка програмного забезпечення (до \_\_.\_\_.2016 р.)

5. Розробка інтерфейсу, планування, тестування розробленої програми (до \_\_.\_\_.201\_ р.)

6. Розробка пояснювальної записки (до \_\_.\_\_.2016 р.).

7. Захист курсової роботи (до \_\_.\_\_.2016 р.).

1.9 Порядок контролю та приймання. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв її оцінювання.

Додаток Б Тексти програмного коду

*Тексти програмного коду програмного забезпечення*

*Знаходження найкоротшого остовного дерева*

(Вид носія даних)

*CD-R*

*12 арк, 35 Кб*

*студента групи ІП-51 І курсу*

Субцельний Олександр Володимирович

**Main.cpp**

#include <QApplication>

#include "graph.h"

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Graph \*wnd = new Graph();

wnd->show();//виклик діалогового вікна

return a.exec();

}

Graph.cpp  
#include "graph.h"

#include "ui\_graph.h"

#include <QMessageBox>

#include "prima.h" // реалізація алгоритму прима

#include "kruskala.h" //реалізація алгоритму Крускала

#include "boruvka.h" //реалізація алгоритму Борувки

#include "WriteRead.h" // запис .dot файлу і вихідного файлу

#include "researchGraph.h" //перевірка на звзність графа

using namespace std;

vector <vertex> graph; //опис графу

vector < pair<int,int> > MST;//остовне дерево

int number\_tree[100];//допоміжний масив для графу

bool different\_edges=true;//перевірка на різність вершин

int n,//к-лькість вершин

m,//к-сть ребер

iter=0;//к-сть ітерацій

unsigned int search\_time;//час роботи алгоритму

void getEdges(){// отримати вершини та уворити допоміжний масив

different\_edges=true;

vector <int> massiv;

int k=0;

bool flag;

for (int i=0;i<graph.size();i++){

flag=true;

for (int j=0;j<massiv.size();j++){

if (graph[i].from==massiv[j])

flag=false;

}

if (flag){

massiv.push\_back(graph[i].from);

number\_tree[graph[i].from]=k++;

}

flag=true;

for (int j=0;j<massiv.size();j++){

if (graph[i].to==massiv[j])

flag=false;

}

if (flag) {

massiv.push\_back(graph[i].to);

number\_tree[graph[i].to]=k++;

}

}

n=massiv.size();

m=graph.size();

for (int i=0;i<n;i++){

if (massiv[i]==massiv.size()){

different\_edges=false;

}

}

}

Graph::Graph(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Graph)//створити діалогове вікно

{

ui->setupUi(this);

ui->okButton->setEnabled(true);

ui->boruvkaButton->setEnabled(false);

ui->primaButton->setEnabled(false);

ui->kruskalaButton->setEnabled(false);

ui->pushButton\_3->setEnabled(false);

ui->pushButton\_4->setEnabled(false);

ui->pushButton\_6->setEnabled(false);

}

void Graph::OkEnabled() //активувати кнопку Enter

{

ui->okButton->setEnabled(true);

}

Graph::~*Graph*() //закрити діалогове вікно

{

delete ui;

}

void Graph::on\_okButton\_clicked() // функція обробки натискання на кнопку Enter

{

vertex temp;

ui->boruvkaButton->setEnabled(true);

ui->primaButton->setEnabled(true);

ui->kruskalaButton->setEnabled(true);

temp.from=ui->fromBox->value();//зчитати вершину виходу

temp.to=ui->toBox->value();//зчитати вершину входу

temp.weight=ui->weightBox->value();//зчитати вагу ребра

graph.push\_back(temp); //записати у граф

}

void Graph::on\_boruvkaButton\_clicked()//фунція обробки натискання на кнопку "Boruvka"

{

getEdges();

if (!research(n,m,graph,number\_tree)) QMessageBox::information(this, "Error", "You Entered non coherent graph.\nPlease reEnter graph"); // Якщо граф не звязний

else if ((research(n,m,graph,number\_tree))&&!different\_edges){

ui->pushButton\_3->setEnabled(true);//розблокувати кнопку "SHOW"

ui->pushButton\_4->setEnabled(true);//розблокувати кнопку "Show Graph"

ui->pushButton\_6->setEnabled(true);//розблокувати кнопку "записати у файл output.txt"

MST = Boruvka(n,m,graph,search\_time,iter);//виклик функції "Boruvka"

}

else QMessageBox::information(this, "Sorry", "This method cant solve this graph.\nChoose Kruskala");

}

void Graph::on\_kruskalaButton\_clicked()//фунція обробки натискання кнопки "Kruskala"

{

getEdges();

if (!research(n,m,graph,number\_tree)) QMessageBox::information(this, "Error", "You Entered non coherent graph.\nPlease reEnter graph");

if (research(n,m,graph,number\_tree)){

ui->pushButton\_3->setEnabled(true);//розблокувати кнопку "SHOW"

ui->pushButton\_4->setEnabled(true);//розблокувати кнопку "Show Graph"

ui->pushButton\_6->setEnabled(true);//розблокувати кнопку "записати у файл output.txt"

MST = Kruskala(n,m,graph,search\_time,iter,number\_tree);//виклик функції "Kruskala"

}

}

void Graph::on\_primaButton\_clicked()

{

getEdges();

if (!research(n,m,graph,number\_tree)) QMessageBox::information(this, "Error", "You Entered non coherent graph.\nPlease reEnter graph");

else if ((research(n,m,graph,number\_tree))&&!different\_edges){

ui->pushButton\_3->setEnabled(true);//розблокувати кнопку "SHOW"

ui->pushButton\_4->setEnabled(true);//розблокувати кнопку "Show Graph"

ui->pushButton\_6->setEnabled(true);//розблокувати кнопку "записати у файл output.txt"

MST = Prima(n,m,graph,search\_time,iter);//виклик функції "Prima"

}

else QMessageBox::information(this, "Sorry", "This method cant solve this graph.\nChoose Kruskala");

}

void Graph::on\_pushButton\_3\_clicked()//обробка натискання кнопки "Show"

{

ui->tableWidget->setRowCount(n-1);//встановити к-сть рядів 1 таблиці

ui->tableWidget->setColumnCount(2);//встановити к-сть стовпців й таблиці

ui->tableWidget->setHorizontalHeaderLabels(QStringList()<<"from"<<"to");//підписати 1 таблицю

for (int i=0;i<ui->tableWidget->rowCount(); i++)

{

QTableWidgetItem \*item\_1 = new QTableWidgetItem(tr("%1").arg(MST[i].first));

QTableWidgetItem \*item\_2 = new QTableWidgetItem(tr("%1").arg(MST[i].second));

item\_1->setBackgroundColor(Qt::gray);

ui->tableWidget->setItem(i,0,item\_1);//заповнити ячейку

ui->tableWidget->setItem(i,1,item\_2);//заповнити ячейку

}

ui->tableWidget\_2->setRowCount(1);//встановити к-сть рядів 2 таблиці

ui->tableWidget\_2->setColumnCount(2);//встановити к-сть стовпців 2 таблиці

ui->tableWidget\_2->setHorizontalHeaderLabels(QStringList()<<"Time[ms]"<<"Iterations");//підписати 2 таблицю

QTableWidgetItem \*item\_1 = new QTableWidgetItem(tr("%1").arg(search\_time));

QTableWidgetItem \*item\_2 = new QTableWidgetItem(tr("%1").arg(iter));

ui->tableWidget\_2->setItem(0,0,item\_1);//заповнити ячейку

ui->tableWidget\_2->setItem(0,1,item\_2);//заповнити ячейку

ui->okButton->setEnabled(false);//заблокувати кнопку

ui->boruvkaButton->setEnabled(false);//заблокувтаи кнопку

ui->primaButton->setEnabled(false);//заблокувати кнопку

ui->kruskalaButton->setEnabled(false);//заблокувти кнопку/

ui->pushButton\_3->setEnabled(false);//заблокуватикнопку

ui->pushButton\_4->setEnabled(true);//розблокуати кнопку

ui->pushButton\_6->setEnabled(true);//розблокувати кнопку

}

void Graph::on\_pushButton\_4\_clicked()

{

createFile(n,m,graph,MST);

system("circo -Tpng -O /home/subtselnyi/Kursova\_Subtselnyi/graph.dot");

system("xdg-open /home/subtselnyi/Kursova\_Subtselnyi/graph.dot.png");

}

void Graph::on\_pushButton\_6\_clicked()//обробка натискання кноки "Write to output.txt"

{

writetoOutput(n, m, MST,search\_time,iter);//запис у файл output.txt

}

void Graph::on\_pushButton\_clicked()//обробка натискання ккнопки "Delete Graph"

{

ui->okButton->setEnabled(true);

ui->boruvkaButton->setEnabled(false);

ui->primaButton->setEnabled(false);

ui->kruskalaButton->setEnabled(false);

ui->pushButton\_3->setEnabled(false);

ui->pushButton\_4->setEnabled(false);

ui->pushButton\_6->setEnabled(false);

while (ui->tableWidget->rowCount() > 0)

{

ui->tableWidget->removeRow(0);//очитити 1 таблцю

}

n=0;m=0;

graph.clear();//знищити опис графу

ui->tableWidget\_2->removeRow(0);//очитити 2 таблицю

}

Boruvka.cpp

#include "boruvka.h"

vector < pair<int,int> > Boruvka(int &n, int &m, vector <vertex> graph,unsigned int &search\_time,int &iter){ //функія реалізації алгоритму Борувки

iter=0;//к-ст ітерацій

vector < pair < int, pair<int,int> > > table (m); //вага - вершина 1 - вершина 2

vector < pair<int,int> > MSTB;//остовне дерево

for (int i=0;i<m;i++){//створення таблиці ваг ребер

iter++;

table[i].second.first = graph[i].from-1;

table[i].second.second = graph[i].to-1;

table[i].first = graph[i].weight;

}

int a[VMax];//допоміжний масив

int colour[VMax];//масив "кольорів"

unsigned int start\_time = clock();//початок відліку часу

sort (table.begin(), table.end());//сортувати таблицю ваг ребер по вазі ребер

iter= iter+ n\*(log(n)/log(2));

int ccount=0;//лічільник

for (int i=0; i<n; i++)//роздати "кольори"

colour[i] = i;

while (ccount!=n-1){//пои не пройшли усі вершнини

iter++;

int k=0;

int l=-1;

vector<bool> used (n,false);//перевірка на пвідвідуваність вершини

while(k!=n-ccount){

iter++;

l++;

if (!used[table[l].second.first]&&(colour[table[l].second.first]!=colour[table[l].second.second])){

used[table[l].second.first]=true;

a[table[l].second.first]=table[l].second.second;

k++;

}

if ((k!=(n-ccount))&&!used[table[l].second.second]&&(colour[table[l].second.first]!=colour[table[l].second.second])){

used[table[l].second.second]=true;

a[table[l].second.second]=table[l].second.first;

k++;

}

}

for (int i=0;i<n;i++){

iter++;

if (colour[i]!=colour[a[i]]){

MSTB.push\_back(make\_pair(a[i]+1,i+1));//додати пару вершин в остовне дерево

int old\_colour = colour[i];

for (int j=0;j<n;j++){

iter++;

if (colour[j]==old\_colour){

colour[j]=colour[a[i]];

}

}

ccount++;

}

}

}

unsigned int end\_time = clock();//кінець роботи алгоритму

search\_time = end\_time - start\_time;

return MSTB;

}

Kruskala.cpp

#include "kruskala.h"

vector < pair<int,int> > Kruskala(int &n,int &m,vector <vertex> graph, unsigned int &search\_time,int &iter, int number\_tree[]){//функція реалізації роботи алгоритму Крускала

iter=0;//кількість ітерацій

vector < pair < int, pair<int,int> > > table (m); // вага - вершина 1 - вершина 2

vector < pair<int,int> > MSTK;

unsigned int start\_time = clock();//початок роботи алгоритму

for (int i=0;i<m;i++){//створення таблиці ваг ребер

iter++;

table[i].second.first = graph[i].from;

table[i].second.second = graph[i].to;

table[i].first = graph[i].weight;

}

sort (table.begin(), table.end());//сортувати за вагою ребра

iter+= n\*(log(n)/log(2));

vector<int> tree\_id (n);//приналежність до дерева

for (int i=0; i<n; i++){//роздати id кожній вершині по приналежності до дерева

tree\_id[i]=i;

}

for (int i=0; i<m; i++){

iter++;

int a = table[i].second.first;

int b = table[i].second.second;

if (tree\_id[number\_tree[a]] != tree\_id[number\_tree[b]]){

MSTK.push\_back (make\_pair (a, b));

int old\_id = tree\_id[number\_tree[b]];

int new\_id = tree\_id[number\_tree[a]];

for (int j=0; j<n; j++){

iter++;

if (tree\_id[j] == old\_id){

tree\_id[j] = new\_id;

}

}

}

}

unsigned int end\_time = clock();//кінець роботи алгоритму

search\_time = end\_time - start\_time;

return MSTK;

}

Prima.cpp

#include "prima.h"

vector < pair<int,int> > Prima(int &n,int &m,vector <vertex> graph,unsigned int &search\_time,int &iter){//функція реалізації алгоритму Прима

iter=0;

vector<bool> used (n,false);//відвідуваність вершин

vector < pair<int,int> > MSTP;

int \*\*table = new int \*[n];//свторити динамічний масив таблиця ваг ребер

for (int i=0;i<n;i++){

table[i] = new int [n];

}

int \*\*min\_dist = new int \*[2];//свторити динамічний масив мінімальна відстань

for (int i=0;i<2;i++){

min\_dist[i] = new int [n];

}

unsigned int start\_time = clock();//початок роботи алгоритму

for (int i=0;i<n;i++)

for (int j=0;j<n;j++)

table[i][j]=INF;

for (int i=0;i<m;i++){

iter++;

table[graph[i].from-1][graph[i].to-1]=graph[i].weight;

table[graph[i].to-1][graph[i].from-1]=graph[i].weight;

}

for (int i=0;i<n;i++)

min\_dist[0][i]=INF;

int start = graph[0].from-1;

int min\_edge,from,to;

used[start]=true;

for (int i=0; i<n; i++) {

iter++;

min\_edge=INF;

for (int j=0;j<n;j++){

iter++;

if (table[start][j]<min\_dist[0][j]){

min\_dist[0][j]=table[start][j];

min\_dist[1][j]=start;

}

if ((min\_edge>min\_dist[0][j])&&used[min\_dist[1][j]]&&!used[j])

{

min\_edge=min\_dist[0][j];

from=min\_dist[1][j];

to=j;

}

}

start=to;

used[to]=true;

MSTP.push\_back (make\_pair (from+1, to+1));//додати пару вершин в остовне дерево

}

unsigned int end\_time = clock();//кінець роботи алгоритму

search\_time = end\_time - start\_time;

for (int i=0;i<n;i++){

delete table[i];

}

delete table;//видалити инамічний масив

for (int i=0;i<2;i++){

delete min\_dist[i];

}

delete min\_dist;//видалити динамічний масив

return MSTP;

}

researchGraph.cpp

#include "researchGraph.h"

bool research(int &n, int &m, vector <vertex> graph,int number\_tree[]){//перевірка на звязність графу

vector <int> tree\_id (n);//приналежність до дерева

for (int i=0;i<n;i++){//роздати id кожній вершині по приналежності до дерева

tree\_id[i]=i;

}

for (int i=0;i<m;i++){

if (tree\_id[number\_tree[graph[i].from]]!=tree\_id[number\_tree[graph[i].to]]){

int old\_id=tree\_id[number\_tree[graph[i].from]];

int new\_id=tree\_id[number\_tree[graph[i].to]];

for (int j=0;j<n;j++){

if (tree\_id[j]==old\_id){

tree\_id[j]=new\_id;

}

}

}

}

int res=tree\_id[0];

for(int i=0;i<n;i++){

if (tree\_id[i]!=res){

return false;//якщо не звязний вернути false

}

}

return true;

}

ReadWrite.cpp

#include "WriteRead.h"

void createFile(int &n, int &m, vector <vertex> graph, vector < pair<int,int> > MST){//сворити файл .dot з описом графу і остовним деревом

char str[5];

FILE \*f;

int MSTweight[VMax];

for (int i=0;i<m;i++){

for (int j=0;j<n-1;j++){

if (((graph[i].from==MST[j].first)&&(graph[i].to==MST[j].second))||((graph[i].from==MST[j].second)&&(graph[i].to==MST[j].first))){

MSTweight[j]=graph[i].weight;

graph[i].weight=inMST;

}

}

}

f=fopen("/home/subtselnyi/Kursova\_Subtselnyi/graph.dot","wt+");

fputs("graph G {",f);

for (int i=0;i<n-1;i++){

sprintf(str, "%d", MST[i].first);

fputs(str,f);

fputs("--",f);

sprintf(str, "%d", MST[i].second);

fputs(str,f);

fputs("[label=",f);

sprintf(str, "%d", MSTweight[i]);

fputs(str,f);

fputs(",penwidth=3];",f);

}

for(int i=0;i<m;i++){

if(graph[i].weight!=inMST){

sprintf(str, "%d", graph[i].from);

fputs(str,f);

fputs("--",f);

sprintf(str, "%d", graph[i].to);

fputs(str,f);

fputs("[label=",f);

sprintf(str, "%d", graph[i].weight);

fputs(str,f);

fputs("];",f);

}

}

fputs("}",f);

fclose(f);

}

void writetoOutput(int &n, int &m, vector < pair<int,int> > MST, unsigned int &search\_time,int &iter){//запис остовного дерева у файл output.txt

char str[5];

FILE \*f;

f=fopen("/home/subtselnyi/Kursova\_Subtselnyi/output.txt","wt+");

fputs("Остовное дерево:\n",f);

for (int i=0;i<n-1;i++){

sprintf(str, "%d", MST[i].first);

fputs(str,f);

fputs("->",f);

sprintf(str, "%d", MST[i].second);

fputs(str,f);

fputs("\n",f);

}

fputs("\n\nЧас виконання алгоритму\n",f);

sprintf(str, "%d", search\_time);

fputs(str,f);

fputs("млс",f);

fputs("\n\nКількість ітерацій алгоритму\n",f);

sprintf(str, "%d", iter);

fputs(str,f);

fclose(f);

}

Boruvka.h

#ifndef BORUVKA\_H

#define BORUVKA\_H

#include "variables.h" //файл з структурою і лібами

#include <math.h>

//файл для реалізації алгоритму Борувки

vector < pair<int,int> > Boruvka(int&, int &, vector <vertex>, unsigned int &,int & );//функія реалізації алгоритму Борувки

#endif // BORUVKA\_H

Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include <QDialog>

//файл для реалізації інтерфейсу

namespace Ui {

class Graph;

}

class Graph : public QDialog

{

Q\_OBJECT

public:

explicit Graph(QWidget \*parent = 0);

~*Graph*();

private:

Ui::Graph \*ui;

private slots: //реалізація кнопок

void OkEnabled();

void on\_okButton\_clicked();

void on\_boruvkaButton\_clicked();

void on\_kruskalaButton\_clicked();

void on\_primaButton\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_6\_clicked();

void on\_pushButton\_clicked();

};

#endif // GRAPH\_H

Kruskala.h

#ifndef KRUSKALA\_H

#define KRUSKALA\_H

#include "variables.h" //файл з структурою і лібами

#include <math.h>

//вайл для реалізації алгоритму Крускала

vector < pair<int,int> > Kruskala(int&,int &, vector <vertex>,unsigned int &,int &, int []);//реалізація алгоритму Крускала

#endif // KRUSKALA\_H

Prima.h

#ifndef PRIMA\_H

#define PRIMA\_H

#include "variables.h" //файл з структурою і лібами

#include <math.h>

//файл для реалізації алгоритму Прима

vector < pair<int,int> > Prima(int&,int &, vector <vertex>,unsigned int &,int &);//реалізація алгоритму Прима

#endif // PRIMA\_H

researchGraph.h

#ifndef RESEARCHGRAPH\_H

#define RESEARCHGRAPH\_H

#include "variables.h" //файл з структурою і лібами

//файл для перевірки звязності графу

bool research(int &, int &,vector <vertex>, int[]);//перевірка на зввязність графа

#endif // RESEARCHGRAPH\_H

WriteRead.h

#ifndef WRITEREAD\_H

#define WRITEREAD\_H

#pragma once

#include "variables.h" //файл з структурою і лібами

//файл для запису в вихідний файл і .dot файл

void createFile(int &n, int &m, vector <vertex> graph, vector < pair<int,int> > MST);//створити .dot файл

void writetoOutput(int &n, int &m, vector < pair<int,int> > MST, unsigned int &,int &iter);//записати у вихыдни файл

#endif // WRITEREAD\_H

Variables.h

#ifndef VARIABLES\_H

#define VARIABLES\_H

//файл для зберігання структури опису графу та лібами

#include <iostream>

#include <vector>

#include <fstream>

#include <stdio.h>

#include <stdlib.h>

#include <iomanip>

#include <algorithm>

#include <ctime>

#define INF 100000

#define inMST 13243546

using namespace std;

struct vertex //структура опису графа

{

int from;

int to;

int weight;

};

const int VMax=100;

#endif // VARIABLES\_H