НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра автоматизованих систем обробки інформації та управління

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

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з                                           «Основи програмування»

(назва дисципліни)

на тему: «Пошук найкоротших шляхів в мережі» .\_

Студента I курсу ІП-63 групи

напряму підготовки 6.050103 «Програмна інженерія».

спеціальності « » .

Бугайова О.О .

.

(прізвище та ініціали)

Керівник Головченко М.М. .

Асистент кафедри АСОІУ .

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ - 2017 рік

Національний технічний університет України “КПІ”

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Програмна інженерія"

Курс 1 Група ІП-63 Семестр 2

**ЗАВДАННЯ**

**на курсову роботу студента**

|  |
| --- |
| Бугайова Олексія Олександровича |

(прізвище, ім’я, по батькові)

|  |  |
| --- | --- |
| 1. Тема роботи | Пошук найкоротших шляхів в мережі |
|  | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи | 6.06.2017 |

|  |  |
| --- | --- |
| 3. Вихідні дані до роботи | Технічне завдання |
|  | |
|  | |
|  | |

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

|  |
| --- |
|  |
|  |
|  |
|  |

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

|  |
| --- |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 6. Дата видачі завдання |  |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи |  |  |
| 2. | Підготовка ТЗ |  |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи |  |  |
| 4. | Розробка алгоритму вирішення задачі |  |  |
| 5. | Узгодження алгоритму з керівником |  |  |
| 6. | Розробка сценарію роботи програми |  |  |
| 7. | Узгодження сценарію роботи програми 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 на перевірку |  |  |
| 16. | Захист курсової роботи |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Бугайов О.О. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | Головченко М.М. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 р.

Анотація

Пояснювальна записка до курсової роботи: 63 сторінки, 20 рисунків, 14 таблиць, 5 посилань.

**Об’єкт дослідження**: задача пошуку найкоротших шляхів в мережі.

**Мета роботи**: дослідження методів пошуку найкоротших шляхів в мережі, створення програмного забезпечення для пошуку найкоротших шляхів у графі, заданому у вигляді матриці суміжності.

Вивчено методи пошуку найкоротших шляхів в мережі. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація алгоритмів Флойда та Данцига.

ГРАФИ , МЕТОД ФЛОЙДА, МЕТОД ДАНЦИГА, ПОШУК НАЙКОРОТШИХ ШЛЯХІВ У МЕРЕЖІ

Зміст

[Вступ 6](#_Toc484547066)

[1 Постановка задачі 7](#_Toc484547067)

[2 Теоретичні відомості 8](#_Toc484547068)

[2.1 Графи 8](#_Toc484547069)

[2.2 Алгоритм Флойда 8](#_Toc484547070)

[2.3 Алгоритм Данцига 10](#_Toc484547071)

[3 Опис алгоритмів 11](#_Toc484547072)

[3.1 Загальний алгоритм 11](#_Toc484547073)

[3.2 Алгоритм Флойда 12](#_Toc484547074)

[3.3 Алгоритм методу Данцига 12](#_Toc484547075)

[4 Опис програмного забезпечення 14](#_Toc484547076)

[4.1 Функціональна схема програмного забезпечення 14](#_Toc484547077)

[4.2 Опис функцій частин програмного забезпечення 14](#_Toc484547078)

[4.2.1 Функції користувача 14](#_Toc484547079)

[4.2.2 Стандартні функції 16](#_Toc484547080)

[5 Тестування програмного забезпечення 19](#_Toc484547081)

[5.1 План тестування 19](#_Toc484547082)

[5.2 Приклади тестування 19](#_Toc484547083)

[5.2.1 Тестування правильності введених значень. 19](#_Toc484547084)

[5.2.1.1 Вхідні дані заповнюються некоректно: символами, від’ємними числами або залишаються пусті клітинки 19](#_Toc484547085)

[5.2.1.2 Назва файлу введена некоректно. 21](#_Toc484547086)

[5.2.2 Правильність роботи алгоритмів 22](#_Toc484547087)

[5.2.2.1 Тестування алгоритму Флойда 22](#_Toc484547088)

[5.2.2.2 Тестування методу Данцига 23](#_Toc484547089)

[5.2.3 Тестування збереження результатів у файл 24](#_Toc484547090)

[5.2.4 Тестування візуалізації графа. 25](#_Toc484547091)

[6 Інструкція користувача 27](#_Toc484547092)

[6.1 Робота з програмою 27](#_Toc484547093)

[6.2 Формат вхідних та вихідних даних 29](#_Toc484547094)

[6.3 Системні вимоги 30](#_Toc484547095)

[7 Аналіз і узагальнення результатів 31](#_Toc484547096)

[7.1 Перевірка результатів роботи програми на достовірність 31](#_Toc484547097)

[7.2 Оцінка ефективності 33](#_Toc484547098)

[Висновки 37](#_Toc484547099)

[Перелік ПОСИЛАННь 38](#_Toc484547100)

[Додаток А Технічне завдання 39](#_Toc484547101)

[Додаток Б Тексти програмного коду 42](#_Toc484547102)

[Лістинг програми (функціональної частини): 43](#_Toc484547103)

[Файл «MainForm.h»: 43](#_Toc484547104)

[Файл «MainForm.cpp»: 56](#_Toc484547105)

[Файл «floid.h»: 56](#_Toc484547106)

[Файл «floid.cpp»: 57](#_Toc484547107)

[Файл «danzig.h»: 58](#_Toc484547108)

[Файл «danzig.cpp»: 58](#_Toc484547109)

[Файл «file.h»: 60](#_Toc484547110)

[Файл «file.cpp»: 60](#_Toc484547111)

Вступ

Як відомо, будь-яку мережу можна представити у вигляді графу. Граф є математичною моделлю найрізноманітніших об'єктів, явищ і процесів, що досліджуються і використовуються в науці, техніці та на практиці. Вони широко використовуються в багатьох сферах науки і техніки

В теорії графів, задача про найкоротший шлях полягає в знаходженні такого шляху між двома вершинами (або вузлами) графу, що сума ваг ребер з яких він складається мінімальна. Прикладом може бути знаходження найкоротшого шляху між двома пунктами на дорожній мапі; в цьому випадку, вершинами є пункти, а ребрами - відтинки дороги із вагами, рівними часу, необхідному для подолання цього відрізку. За таким же принципом можна предствити будь-яку мережу, як орієнтовний зважений граф.

Для того, щоб уявити нвіщо ж потрібен цей тип задач, приведемо приклад із сьогодення. Усі GPS розглядають мережу доріг, як граф, де перехрестя – вершини, а прямі ділянки дороги – ребра, тож за допомогою таких алгоритмів, можна допомогти людині знайти найкоротший шлях між 2 точками простору.

Але якщо уявити, що роботу по знаходженню найкоротших шляхів має виконувати людина, то стає зрозуміло, що на це потрібно витратити багато часу, тому для таких задач використовують комп’ютери

У цій роботі для знаходження найкоротших відстаней використовувалися алгоритм Флойда та Данцига, також було розроблено відповідне програмне забезпечення та наданий аналіз методам, що розглядалися.

# Постановка задачі

Розробити програмне забезпечення, яке вирішує задачу пошуку найкоротшого шляху в графі, розмірність якого задається керівником, заданому у вигляді переліку ребер, які задає керівник, або зчитуються з файлу, наступними методами:

1. Метод Флойда;
2. Метод Данцига.

Вхідні дані: кількість вершин, та ребер, що обираються користувачем, вершини графу між якими потрібно шукати найкоротший шлях, перелік ребер, які вводяться користувачем, або зчитуються з файлу та вибраний користувачем алгоритм пошуку найкоротших шляхів.

Вихідні дані: візуалізація даного графа. Матриця суміжності, матриця відстней, матриця предків, а також найкоротший шлях між обрними вершинми, всі ці данні зберігаються у файл.

# Теоретичні відомості

## **Графи**

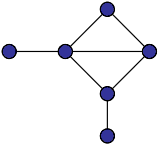
Граф — це сукупність об'єктів із зв'язками між ними. Об'єкти розглядаються як вершини, або вузли графу, а зв'язки — як дуги, або ребра. Для різних областей використання види графів можуть в ідрізнятися орієнтованістю, обмеженнями на кількість зв'язків і додатковими даними про вершини або ребра.

Рисунок 2.1 – Зображення графа

Граф, ребра якого не мають напряму називається — неорієнтованим, в іншому випдку — орієнтованим. Якщо пара вершин сполучається кількома ребрами чи дугами одного напрямку, то такі ребра називають кратними. Ребро, що сполучає вершину саму із собою називається петлею. Граф без кратних дуг і петель називається простим.

Вершини, сполучені ребром чи дугою, називають суміжними, також називають суміжними ребра, що мають спільну вершину

Матриця суміжності це спосіб представлення графа у вигляді матриці розміром V\*V (де V кількість вершин). При цьому дорівнює 0, якщо вершини i та j не є суміжними, та 1 для суміжних вершин. Для зваженого графа дорівнює вазі ребра (*i, j*).

## **Алгоритм Флойда**

Ключова ідея алгоритму - розбиття процесу пошуку найкоротших шляхів на фази**.**

Перед *k*-ою фазою (*k=1…n*) вважається, що в матриці відстаней *d*[][] збережені довжини таких найкоротших шляхів, які містять в якості внутрішніх вершин тільки вершини з безлічі {1, 2,…, *k*-1} (Вершини графа ми нумеруем, починаючи з одиниці).

Іншими словами, перед *k* -ою фазою величина *d*[][] дорівнює довжині найкоротшого шляху з вершини *i* в вершину *j*, Якщо цим шляхом дозволяється заходити тільки в вершини з номерами, меншими *k* (Початок і кінець шляху не вважаються).

Легко переконатися, що щоб це властивість виповнилося для першої фази, досить в матрицю відстаней *d*[][] записати матрицю суміжності графа: *d*[*i*][j]=*g*[i][j] - вартості ребра з вершини *i* в вершину *j* . При цьому, якщо між якимись вершинами ребра немає, то записати слід величину "нескінченність". З вершини в саму себе завжди слід записувати величину 0 , Це критично для алгоритму.

Нехай тепер ми перебуваємо на *k* -ій фазі, і хочемо перерахувати матрицю *d*[][] таким чином, щоб вона відповідала вимогам вже для *k + 1*-ої фази. Зафіксуємо якісь вершини *i* і *j* . У нас виникає два принципово різних випадку:

1. Найкоротший шлях з вершини *i* в вершину *j* , Якому дозволено додатково проходити через вершини {1, 2,…, *k*}, збігається з найкоротшим шляхом, яким дозволено проходити через вершини множини {1, 2,…, *k*-1}. У цьому випадку величина *d*[*i*][j] не зміниться при переході з *k* -ої на *k+1* - у фазу.
2. "Новий" найкоротший шлях став краще "старого" шляху. Це означає, що "новий" найкоротший шлях проходить через вершину *k* . Відразу відзначимо, що ми не втратимо спільності, розглядаючи далі тільки прості шляхи (тобто шляху, що не проходять по якійсь вершині двічі).Тоді зауважимо, що якщо ми розіб'ємо цей "новий" шлях вершиною *k* на дві половинки (одна йде *i - k* , а інша *k - j*), то кожна з цих половинок вже не заходить в вершину *k*. Але тоді виходить, що довжина кожної з цих половинок була порахована ще на *k - 1* -ій фазі або ще раніше, і нам достатньо взяти просто суму  *d*[*i*][*k*]+*d*[*k*][j], вона і дасть довжину "нового" найкоротшого шляху.

Об'єднуючи ці два випадки, отримуємо, що на *k* -ій фазі потрібно перерахувати довжини найкоротших шляхів між усіма парами вершин *i* і *j* наступним чином:

(2.1)

Таким чином, вся робота, яку потрібно зробити на *k*-ій фазі - це перебрати всі пари вершин і перерахувати довжину найкоротшого шляху між ними. В результаті після виконання *n* фази в матриці відстаней *d*[*i*][*j*] буде записана довжина найкоротшого шляху між *i* і *j*, або "нескінченність", якщо шляху між цими вершинами не існує.

Останнє зауваження, яке слід зробити, - то, що можна не створювати окрему матрицю *new\_d*[][] для тимчасової матриці найкоротших шляхів на *k*-ій фазі: всі зміни можна робити відразу в матриці *d*[][]. Справді, якщо ми поліпшили (зменшили) якесь значення в матриці відстаней, ми не могли погіршити тим самим довжину найкоротшого шляху для якихось інших пар вершин, оброблених пізніше.

Асимптотика алгоритму, очевидно, становить O(N ^ 3).

## **Алгоритм Данцига**

Алгоритм для знаходження найкоротших шляхів до всіх вершин планарного спрямованого графа. Названий на честь американського математика Джорджа Данцига. Алгоритм близький до алгоритму Флойда, відрізняється від нього лише іншим порядком виконання одних і тих же операцій.

Крок 1

Пронумерувати вершини вихідного графа цілими числами від 1 до N. Сформувати матрицю *D0* (розмірністю *NxN*), кожен елемент *i, j* якої *d0ij* визначає довжину найкоротшої дуги яка веде з вершини *i* у вершину *j*. В разі відсутності такої дуги покласти *d0ij = ∞.* Крім того, для всіх і і m покласти *dmii = 0*.

Крок 2

Тут через *Dm* позначається матриця розмірністю *mxm* з елементами *dmij, m = 1,2,...,N.* Послідовно визначити елементи матриці *Dim* з елементів матриці *Dm-1* для *m* що приймають значення *1,2,…N.*

(2.1)

(2.2)

(2.3)

Асимптотика алгоритму, очевидно, становить O(N ^ 3).

# Опис алгоритмів

## **Загальний алгоритм**

1. ПОЧАТОК
2. ЯКЩО обраний пункт «Зчитати з файлу», ТО ввести назву файлу, перейти до п.6.
3. ЯКЩО була змінена кількість вершин, ТО змінити відповідно новому значенню максимальне значення ребер, ЯКЩО користувач нічого не обрав наступні дії будуть відбуватися з даними за замовчуванням.
4. ЯКЩО змінена кількість ребер, ТО додати у таблицю ребер кількість записів, що дорівнює кількості введених ребер. І заповнити їх відповідним чином:
   1. ЯКЩО деякі ребра не були введені, або були введені некоректні дані, ТО вивести повідомлення про помилку.
5. ЯКЩО був обраний початок і кінець маршруту:
   1. ЯКЩО елемент не виходить за діапазон дозволених значень, ТО змінити дані наявні в полі на введені, ІНАКШЕ залишити раніше введені дані в цьому полі.
6. ЯКЩО натиснута кнопка «Розрахувати», ТО:
   1. ЯКЩО введені дані не пройшли перевірку, ТО вивести відповідне повідомлення про помилку і припинити роботу.
   2. ЯКЩО обраний пункт «Алгоритим Флойда», ТО обчислити результат методом Флойда.
   3. ЯКЩО обраний пункт «Алгоритим Данцига», ТО обчислити результат методом Данцига.
   4. ЯКЩО не був обраний жодний пункт, ТО вивести повідомлення про помилку, повернутися до п.6.
7. Вивести результати роботи програми у вікно і у файл.
8. ЯКЩО користувач бажає продовжити роботу, ТО повернутися до п.2.
9. КІНЕЦЬ

## **Алгоритм Флойда**

Усі змінні що використовувались при описі даного алгоритму та їх призначення описані в таблиці 3.1.

Таблиця 3.1 – Змінні та їхні призначення для алгоритму Флойда

|  |  |
| --- | --- |
| Змінна | Призначення |
| N | Кількість вершин графа |
| lMatrix | Матриця відстаней (), що містить початкові значення відстаней між вершинами |
| fMatrix | Матриця предків (), що містить номер вершини у відповідних стовпцях |
| sMatrix | Матриця суміжності (), що початкові ваги ребер |
| INF | Велике число, для позначення відсутності шляху |

Алгоритм:

1. ПОЧАТОК
2. Ініціалізація lMatrix і fMatrix за допомогою sMatrix.
3. ЦИКЛ проходу по обхідних вершинах *k* (*k* = 0, N):
   1. ЦИКЛ проходу по вершинах *i* (*i* = 0, N):
      1. ЦИКЛ проходу по вершинах *j* (*j* = 0, N):
         1. ЯКЩО lMatrix[*i*][*j*] > lMatrix[*i*][*k*] + lMatrix[*k*][*j*], ТО:
            1. lMatrix[*i*][*j*] = lMatrix[*i*][*k*] + lMatrix[*k*][*j*].
            2. fMatrix[*i*][*j*] = fMatrix[*i*][*k*].
4. КІНЕЦЬ

## **Алгоритм Данцига**

Усі змінні, що використовувались при описі даного алгоритму та їх призначення описані в таблиці 3.2.

Таблиця 3.2 – Змінні та їхні призначення для алгоритму Данцига

|  |  |
| --- | --- |
| Змінна | Призначення |
| N | Кількість вершин графа або розмірність матриці суміжності |
| lMatrix | Матриця відстаней (), що містить початкові значення відстаней між вершинами |
| fMatrix | Матриця предків (), що містить номер вершини у відповідних стовпцях |
| sMatrix | Матриця суміжності (), що початкові ваги ребер |
| INF | Велике число, для позначення відсутності шляху |

Алгоритм:

1. ПОЧАТОК
2. Ініціалізація lMatrix і fMatrix за допомогою sMatrix.
3. ЦИКЛ проходу по обхідних вершинах *m* (*m* = 0, N):
   1. ЦИКЛ проходу по вершинах *i* (*i* = 0, *m*):
      1. ЦИКЛ проходу по обхідних вершинах *j* (*j* = 0, *m*):
         1. ЯКЩО lMatrix[*i*][*m*] > lMatrix[*j*][*m*] + lMatrix[*i*][*j*], ТО:
            1. lMatrix[*i*][*m*] = lMatrix[*j*][*m*] + lMatrix[*i*][*j*].
            2. fMatrix[*i*][*m*] = fMatrix[*i*][*j*].
   2. ЦИКЛ проходу по вершинах *j* (*i* = 0, *m*):
      1. ЦИКЛ проходу по обхідних вершинах *i* (*i*= 0, *m*):
         1. ЯКЩО lMatrix[*m*][*j*] > lMatrix[*m*][*i*] + lMatrix[*i*][*j*], ТО:
            1. lMatrix[*m*][*j*] = lMatrix[*m*][*i*] + lMatrix[*i*][*j*].
            2. fMatrix[*m*][*j*] = fMatrix[*m*][*i*].
   3. ЦИКЛ проходу по вершинах *i* (*i* = 0, *m*):
      1. ЦИКЛ проходу по вершинах *j* (*j* = 0, *m*):
         1. ЯКЩО lMatrix[*i*][*j*] > lMatrix[*i*][*m*] + lMatrix[*m*][*j*], ТО:
            1. lMatrix[*i*][*j*] = lMatrix[*i*][*m*] + lMatrix[*m*][*j*].
            2. fMatrix[*i*][*j*] = fMatrix[*i*][*m*].
4. КІНЕЦЬ

# Опис програмного забезпечення

## **Функціональна схема програмного забезпечення**

Дана програма містить: 4 файли реалізації, а також 4 заголовні файли і зображена на рисунку 4.1.



Рисунок 4.1 – Функціональна структура програмного забезпечення

## **Опис функцій частин програмного забезпечення**

Програма поділена на наступні модулі:

1. dancig.h – реалізує алгоритм Данцига;
2. floid.h – реалізує алгоритм Флойда;
3. MainForm.h – заголовочний файл “Windows Forms” який відповідє за візульну частину програми;
4. file.h – відповідає за функції, пов’язані із введеням/виведенням у файл.

### Функції користувача

Всі користувацькі функції, описані в таблиці 4.1.

Таблиця 4.1 — Користувацькі функції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1. | *min* | Поверненя мінімльного значення | 2 числа, які потрібно порівняти | Мінімальне число серед 2 вхідних | *floid.h*  *danzig.h* |
| 2. | *scanMatrix* | Зчитування матриці суміжності | Розмірність, матриця суміжності | - | *MainForm.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 3. | *printGraph* | Візуалізція графа | Матриця суміжності, кількість вершин | - | *MainForm.h* |
| 4. | *printMatrix* | Виводить матрицю (NxN) у вікно | Матриця, розмірність | - | *MainForm.h* |
| 5. | *showPath* | Відновлення найкоротшого шляху | Початок, кінець, матриця предків і відстней | - | *MainForm.h* |
| 6. | *checkGraph* | Перевірка введених даних перед зчитуванням | - | True у випадку корректності даних, інкше False | *MainForm.h* |
| 7. | *edgesNumber\_ValueChanged* | Обробка зміни значення кількості ребер | - | - | *MainForm.h* |
| 8. | *calculate\_Click* | Обробка зміни натискання на кнопку «Розрахувати» | - | - | *MainForm.h* |
| 9. | *inputEdges\_*  *CellValidated* | Перевірка значень введених у таблицю ребер | - | - | *MainForm.h* |
| 10. | *vertexNumber\_ValueChanged* | Обробка зміни значення кількості вершин | - | - | *MainForm.h* |
| 11. | *getFile\_CheckedChanged* | Обробка вибору типу зчитування графа | - | - | *MainForm.h* |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 12. | *floid* | Пошук найкоротших шляхів алгоритмом Флойда | Матриця суміжності, кількість вершин, порожня матриця відстаней, предків | - | *floid.h* |
| 13. | *danzig* | Пошук найкоротших шляхів алгоритмом Данцига | Матриця суміжності, кількість вершин, порожня матриця відстаней, предків | - | *danzig.h* |
| 14. | *getGraph*  *FromFile* | Зчитування графу з файлу | Порожня матриця суміжності, ім’я файлу, змінні для збереження: вершин, ребер, початку та кінця маршруту | True якщо вдалося зчитати, False в іншому випадку | *file.h* |
| 15. | *saveResults* | Збереження результатів роботи програми у файл | Ім’я файлу, матриця суміжності, відстаней, предків, розмірність, початок, кінець | True якщо вдалося зберегти, False в іншому випадку | *file.h* |

### Стандартні функції

Всі стандартні функції, використані курсовій роботі, описані в таблиці 4.2.

Таблиця 4.2 — Стандартні функції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | *System::Convert::ToString* | Перетворення значення на рядок | Значення, що потрібно перетворити | рядок | *stdafx.h* |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 2 | *System::Convert::ToInt32* | Перетворення на int | Змінна простого типу | Ціле число | *stdafx.h* |
| 3 | *DrawEllipse* | Малює еліпс | Пара координат, та ширина і висота прямокутника описаного навколо | - | *stdafx.h* |
| 4 | *DrawString* | Малює вказаний рядок | Пара координат та ширина і висота прямокутника описаного навколо, шрифт, рядок | - | *stdafx.h* |
| 5 | *DrawLine* | Малює лінію | Координати між якими потрібно намалювати відрізок, параметри пера для малюнка | - | *stdafx.h* |
| 8 | *is\_open* | Перевіряє чи існує файловий поток | - | True, False | *fstream* |
| 9 | *getline* | Записує рядок з файлового потоку у вказаний рядок | Файловий поток  Рядок в який записується інформація | - | *fstream* |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 10 | *close* | Закриває файловий поток | Файловий поток | - | *fstream* |
| 11 | *time* | Повертає час від 1 січня 1970  (сек) | Покажчик для зберігання значення часу | Повертає час від 1 січня 1970 (сек) | *Time.h* |

# Тестування програмного забезпечення

## **План тестування**

1. Тестування правильності введених значень:
   1. Вхідні дані заповнюються некоректно (символами, великими числами, від’ємними числами або залишаються пусті клітинки);
   2. Введене ім’я неіснуючого файлу;
2. Тестування правильності роботи алгоритмів Данцига та Флойда:
   1. Тестування алгоритму Флойда;
   2. Тестування алгоритму Данцига;
3. Тестування збереження результатів у файл.
4. Тестування візуалізації графа.

## **Приклади тестування**

### **Тестування правильності введених значень.**

#### Вхідні дані заповнюються некоректно: символами, від’ємними числами або залишаються пусті клітинки

(Таблиця 5.1 — Введені некоректні дані; Рисунок 5.1 — Введені некоректні дані; 5.2 — Введені від’ємні числа; 5.3 — Залишені пусті поля).

Таблиця 5.1— Введені некоректні дані

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | «фів» «-1» «» «1000000» |
| Схема проведення тесту | Введення неправильних даних у текстові поля |
| Очікуваний результат | Повідомлення про введення некоректих даних |
| Стан програми після проведення випробувань | Видано помилку «Ви ввели невірні дані» |

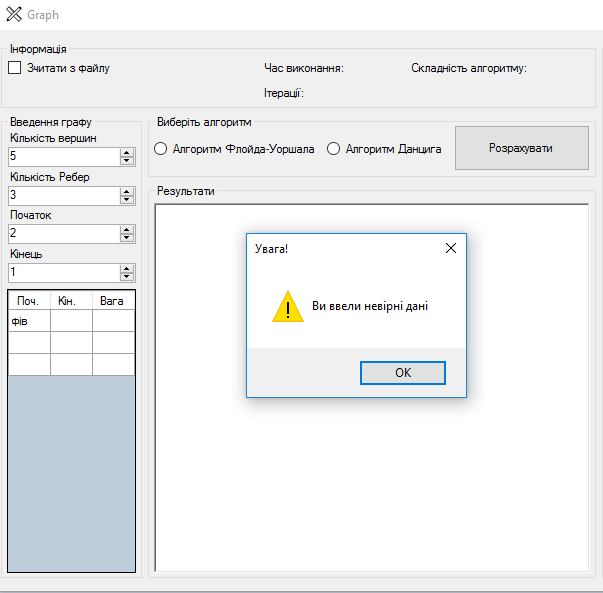


Рисунок 5.1— Введені некоректні дані

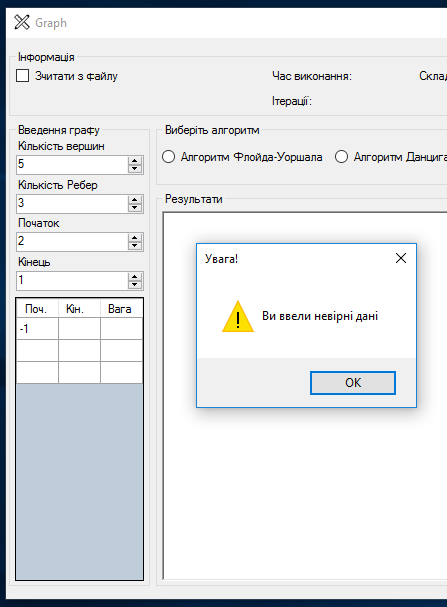


Рисунок 5.2— Введені від’ємні числа

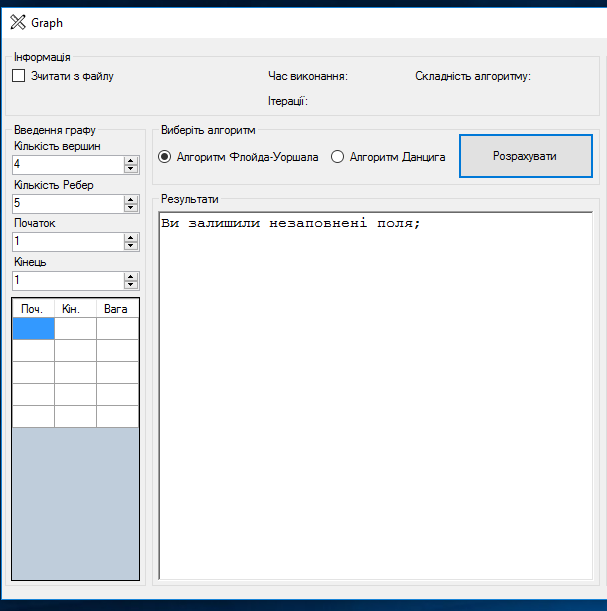


Рисунок 5.3 — Залишені пусті поля

#### Назва файлу введена некоректно.

(Таблиця 5.2 — Некоректна назва файлу; Рисунок 5.4 — Некоректна назва файлу).

Таблиця 5.2 — Некоректна назва файлу

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректної назви файлу |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | «Неіснує» |
| Схема проведення тесту | Ввід назви файла у відповідне поле |
| Очікуваний результат | Повідомлення про помилку  зчитування |
| Стан програми після проведення випробувань | Видано помилку «Помилка під час зчитування файлу» |

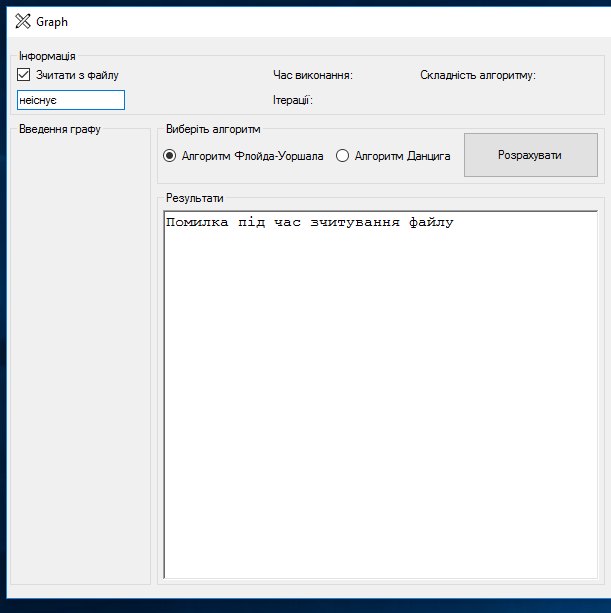


Рисунок 5.2 — Некоректна назва файлу

### **Правильність роботи алгоритмів**

#### Тестування алгоритму Флойда

При введенні коректних вхідних даних метод Флойда виводить правильний результат (Таблиця 5.3 — Коректність алгоритму Флойда та Рисунок 5.5 — Правильність роботи алгоритму Флойда).

Таблиця 5.3 — Коректність алгоритму Флойда

|  |  |
| --- | --- |
| Мета тесту | Перевірити роботу метода Флойда при коректних вхідних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Граф з файлу |

Таблиця 5.3 — Продовження

|  |  |
| --- | --- |
| Схема проведення тесту | Введення назви файлу у відповідне поле, та запуск програми |
| Очікуваний результат | Правильне виконання алгоритму Флойда та коректні вихідні дані |
| Стан програми після проведення випробувань | Алгоритм знайшов відповідний найкоротший шлях. |

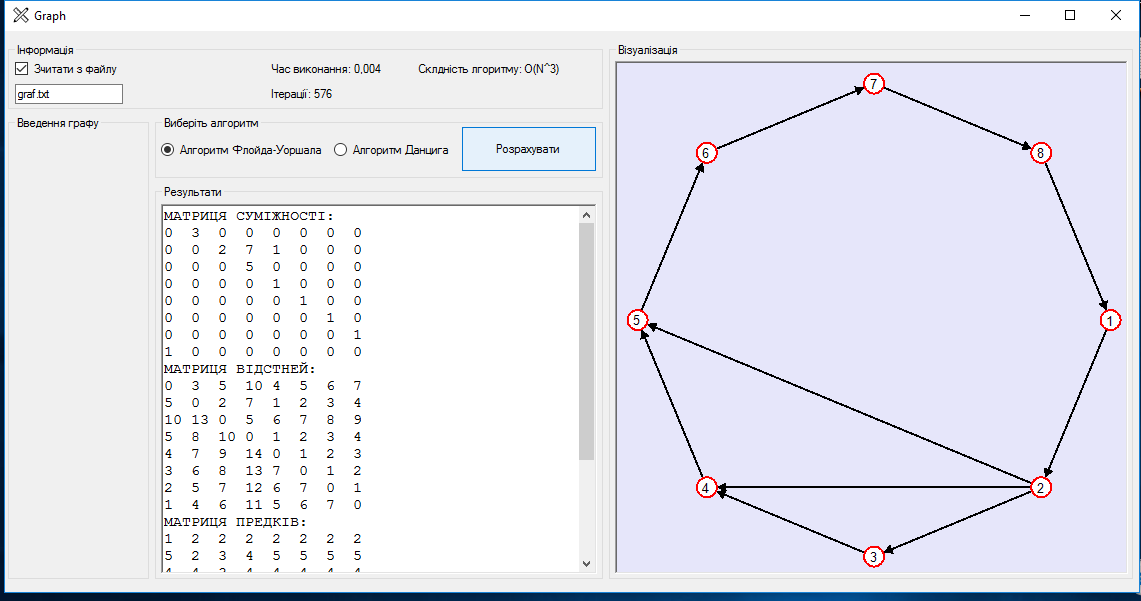


Рисунок 5.5 — Правильність роботи алгоритму Флойда

#### Тестування методу Данцига

При введенні коректних вхідних даних метод Данцига виводить правильний результат (Таблиця 5.4 — Коректність алгоритму Данцига та Рисунок 5.6 — Правильність роботи алгоритму Данцига).

Таблиця 5.4— Коректність алгоритму Данцига

|  |  |
| --- | --- |
| Мета тесту | Перевірити роботу алгоритму Данцига при коректних вхідних даних |
| Початковий стан програми | Відкрите вікно програми |

Таблиця 5.4 — Продовження

|  |  |
| --- | --- |
| Вхідні дані | Файл зі структурою графа |
| Схема проведення тесту | Введення файлу у відповідне поле, та запуск програми |
| Очікуваний результат | Правильне виконання алгоритму Данцига та коректні вихідні дані |
| Стан програми після проведення випробувань | Алгоритм знайшов відповідний найкоротший шлях. |

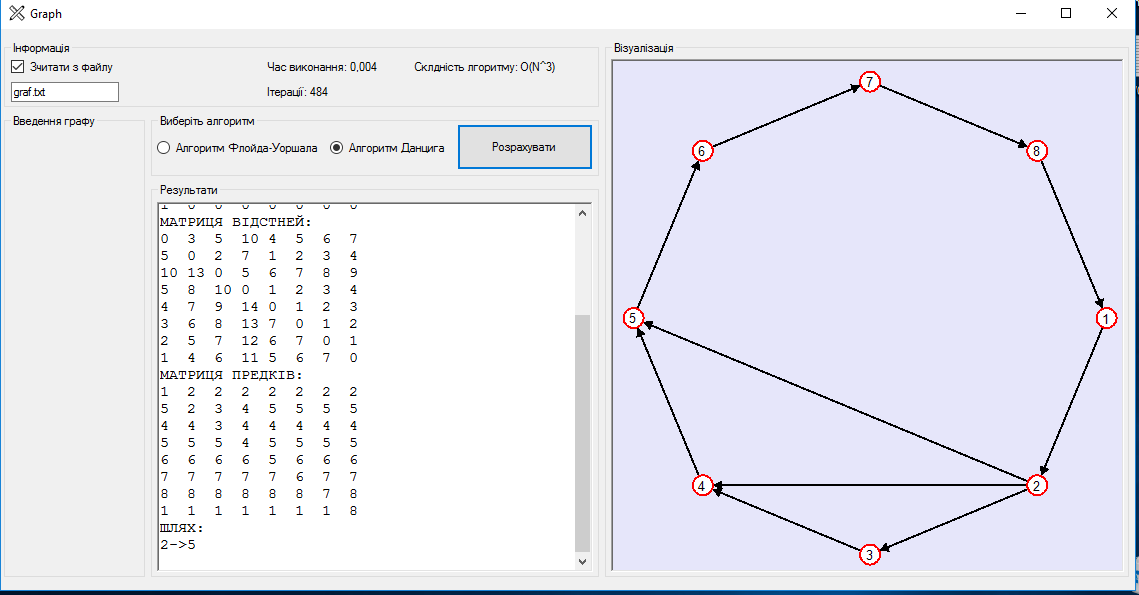


Рисунок 5.6 — Правильність роботи алгоритму Данцига

### **Тестування збереження результатів у файл**

Усі данні роботи програмии зберігаються у файл «results.txt», який знходиться у папці проекту (Таблиця 5.5 — Збереження у файл та Рисунок 5.7 — Збереження у файл).

Таблиця 5.5 — Збереження у файл

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність зберігання результатів роботи прогрми у файл |

Таблиця 5.5 — Продовження

|  |  |
| --- | --- |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Файл зі структурою графа |
| Схема проведення тесту | Введення назви файла у відповідне поле, та запуск програми |
| Очікуваний результат | Правильно заповнений файл «results.txt», який свідчить про правильність роботи функції зберігання |
| Стан програми після проведення випробувань | Данні були правильно записані у файл. |

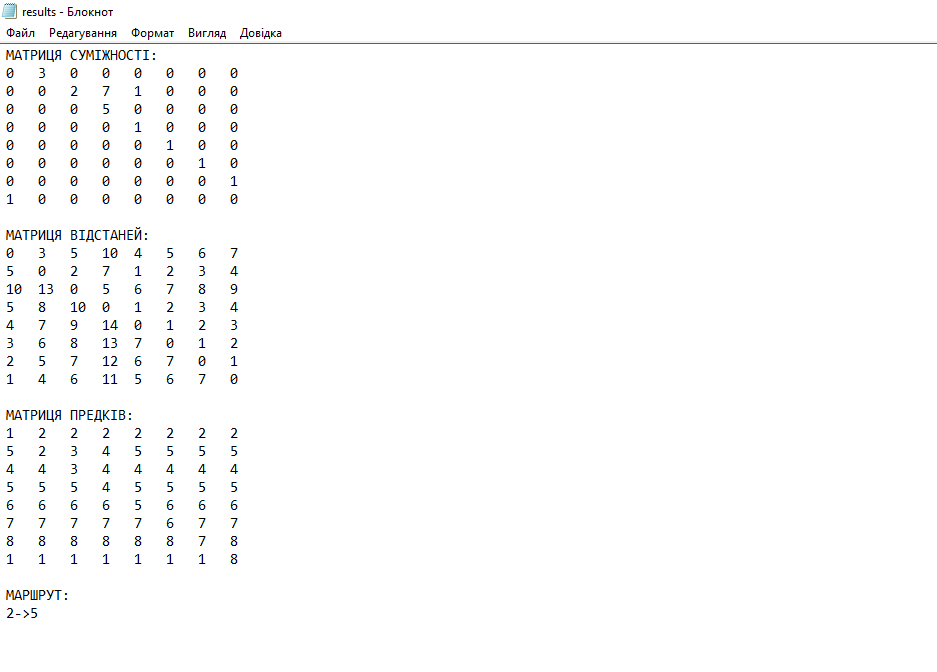


Рисунок 5.7 — Збереження у файл

### **Тестування візуалізації графа.**

Перевірка чи співпадає зображений граф із введеним. (Таблиця 5.6 — Візуалізація графа та Рисунок 5.8 — Візуалізація графа).

Таблиця 5.6 — Візуалізація графа

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність візулізації введеного графа |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Назва файла, який містить структуру графа |
| Схема проведення тесту | Ввести назву вхідного файлу у відповідне поле, і запустити програму |
| Очікуваний результат | Правильно зображений граф |
| Стан програми після проведення випробувань | Програма виконала розрахунки і првильно зобразила граф |

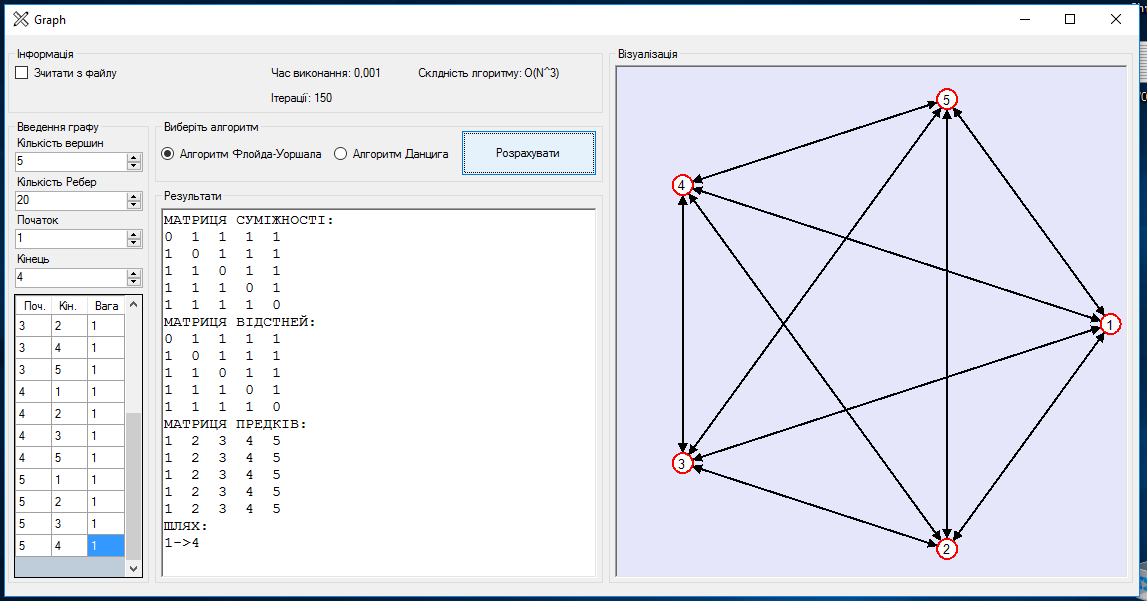


Рисунок 5.7 — Візуалізація графа

# Інструкція користувача

## **Робота з програмою**

Після запуску програми (подвійний клік по іконці файлу з розширенням \*.exe), відкривається головне вікно програми (Рисунок 6.1 – Головне вікно програми).

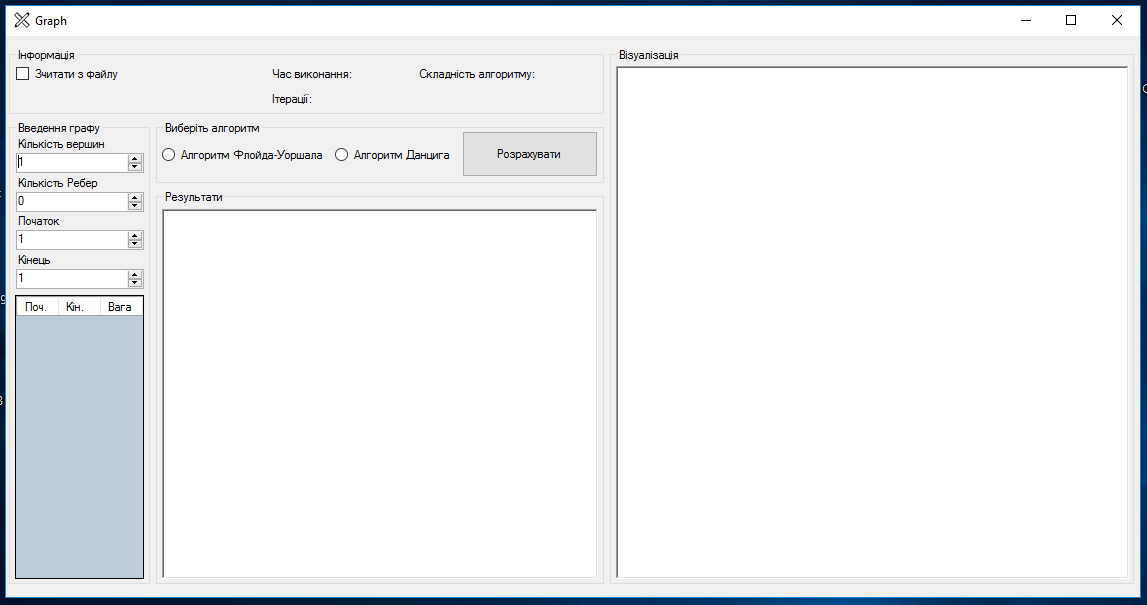


Рисунок 6.1 – Головне вікно програми

Після запуску програми, користувач повинен обрати метод задання графу. Якщо користувач бажає зчитати граф з файлу, то необхідно обрати пункт «Зчитати з файлу» та ввести у відповідне поле назву вхідного файлу. В іншому випадку користувачу потрібно обрати самому структуру графа. Обрати кількість вершин, ребер. Обрати початок та кінець маршруту, який необхідно знайти, та заповнити таблицю ребер, де 1 колонка відповідає за початок ребра, друга за кінець, а третя за вагу ребра.

Далі потрібно обрати, яким алгоритмом програмі робити розрахунки (Поле «Вибір алгоритму»).

Наприклад, спробуємо знайти найкоротший шлях у графі із 10 вершинами, та 12 ребрми між вершинами 4 та 1. При цьому структуру графа будемо вводити вручну. І використаємо алгоритм Флойда.

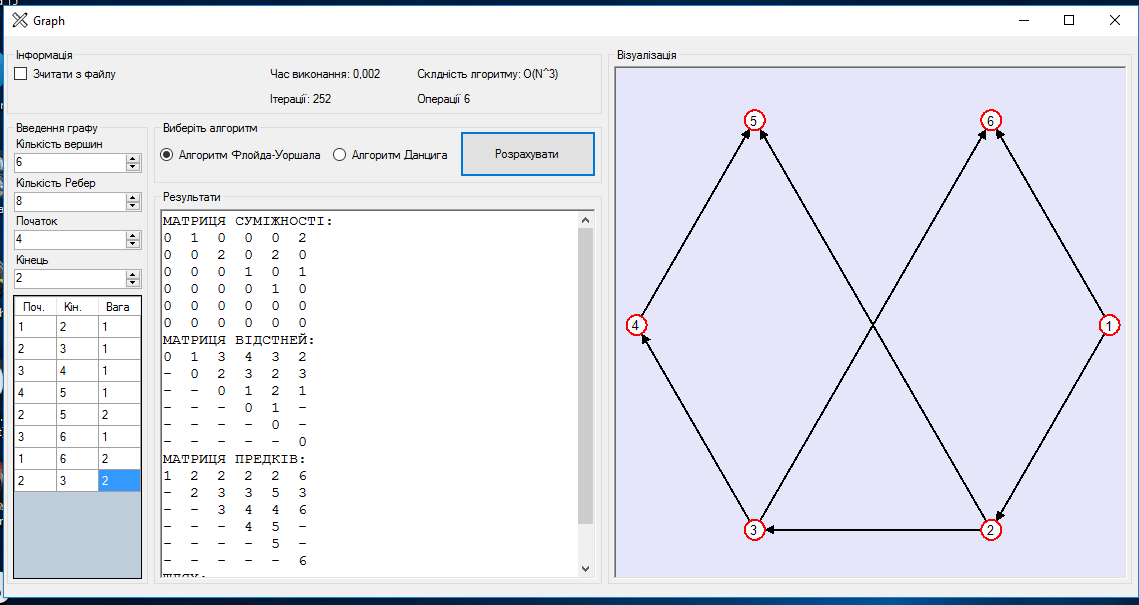


Рисунок 6.2 – Вікно програми після виконання всіх операцій

Як бачимо (Рисунок 6.2 – Вікно програми після виконання всіх операцій), програма правильно зобразила граф, та отримала правильний шлях між вершиною 4, та 2

Окрім того програма виводить додаткову інформацію про роботу алгоритму, а саме роботу у (мкс), кількість ітерацій, та елементарних операцій.

Також програма виводить усі результати у окремий файл «results.txt»

(Рисунок 6.3 –Результат виконання програми записаний у файл).

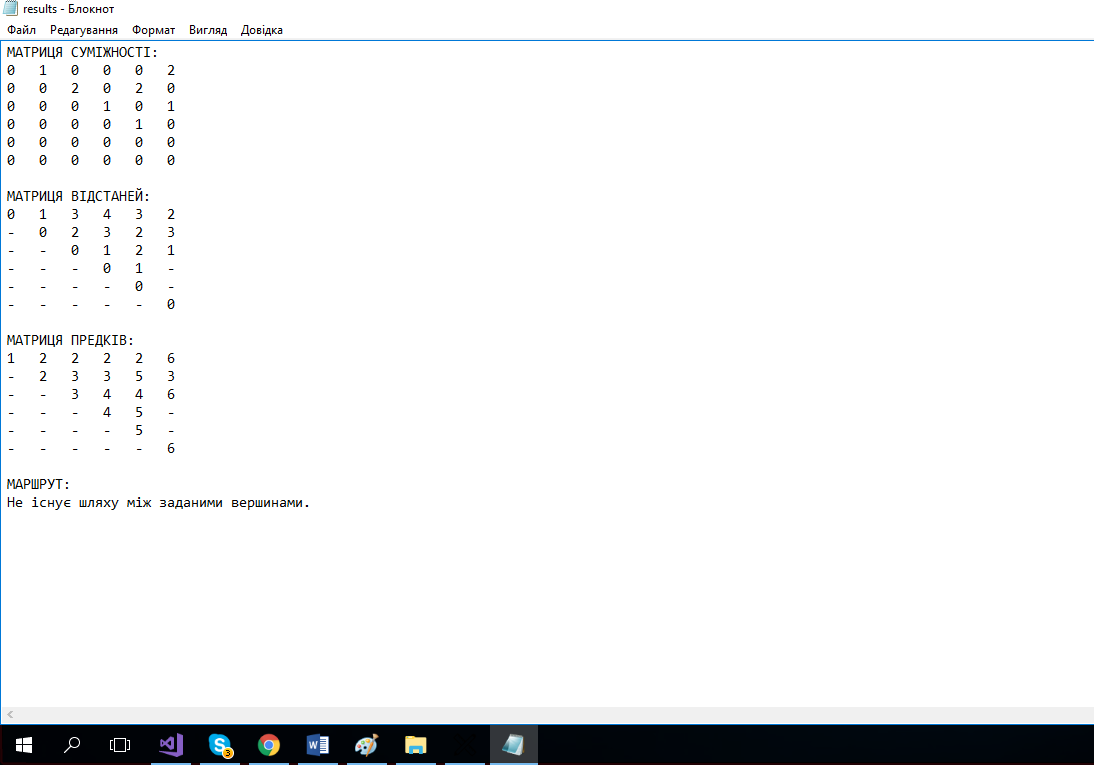


Рисунок 6.3 – Результат виконання програми записаний у файл

Для продовження роботи слід ввести нові данні та повторити описані дії. Для завершення роботи з програмою треба натиснути на хрестик у правому верхньому куті.

## **Формат вхідних та вихідних даних**

У поле «Кількість вершин графа» приймаються цілі числа. Інші символи вписати з клавіатури неможливо, але їх можна вставити з контекстного меню. В такому випадку значення цього поля буде автоматично змінено на минуле значення. Введені числа, більше за 20, будуть автоматично змінені на 20. Стрілками виставити числа не з діапазону 1 – 20 неможливо.

У поле «Кількість ребер» можна вводити кількість ребер із діапазону, який залежить від кількості вершин, наприклад для 3 вершин можна ввести 6 ребер.

У поле назва файлу можна вводити будь-які символи, проте якщо такого файла не існуватиме, прогрма виведе повідомлення про помилку.

У полях «Початок», «Кінець» вводяться спочатку вершина з якої шукати шлях, а потім до якої шукати. Неможливо ввести вершини, яких немає в графі.

Далі іде таблиця заповнення ребер, у першому стовпчику потрібно ввести почток ребра, у другому кінець ребра, а у 3 – його вагу. У цьому полі відбувається перевірка на ніснуючі вершини, а також нечисельні дані.

У полі «Інформація» виводиться інформація про роботу лгоритмів. Час виконання (мкчс), що є дійсним числом. Операції, ітерації, що є цілими числами.

Результат виконання програми: матриця суміжності, відстней, предків, шлях обходу - заносяться у текстовий файл.

## **Системні вимоги**

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 | Windows 7/ Windows 8/Windows 10 |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
|  | Мінімальні | Рекомендовані |
| Оперативна пам'ять | 256 MB RAM | 1 GB RAM |
| Відеоадаптер | Intel GMA 950 або будь-який інший з відеопам'яттю об'ємом не менше 64 МБ | |
| Дисплей | 1280x720 | 1920x1024 |
| Прилади вводу | Клавіатура, комп’ютерна миша | |

# Аналіз і узагальнення результатів

Метою цієї курсової роботи була розробка програмного збезпечення, яке б вирішувло проблему пошуку найкоротшого шляху в мережі за допомогою алгоритму Данцига та Флойда. У процесі тестування цих алгоритмів, стало зрозуміло, що алгоритм Данцига пряцює дещо швидше ніж алгоритм Флойда, що пов’язано з іншим порядком операцій.

Дана програма успішно справляється з поставленою задачею, і знаходить завжди правильні найкоротші шляхи, якщо такі існують.

Під час тестування програми, були виявлені усі її слабкі місця, та були усунуті. Тобто програма правильно реагує на всі неоднозначні ситуації, і в таких ситуаціях завжди повідомляє користувача про помилки.

## Перевірка результатів роботи програми на достовірність

Для перевірки результатів скористаємося відомим сервісом graphonline. Спочатку перевіримо на правильність алгоритм Флойда.

а) алгоритм Флойда.

Результат роботи алгоритму Флойда наведено на рисунку 7.1

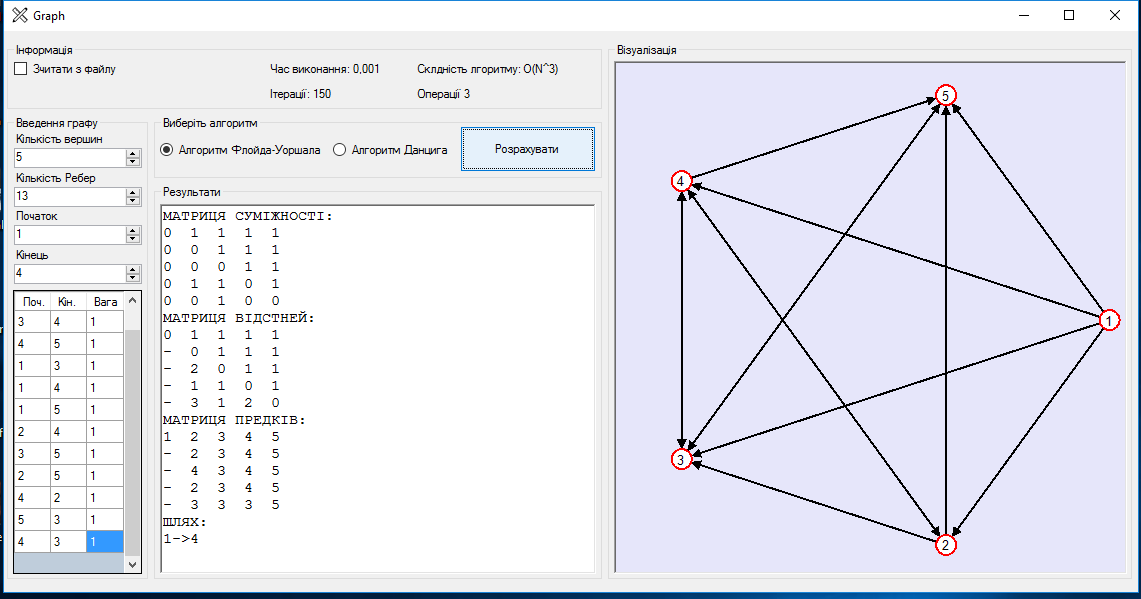


Рисунок 7.1 – Результат виконання алгоритму Флойда

Результати роботи програми відповідають результатам роботи graphonline (Рисунок 7.2), тож можна зробити висновок, що алгоритм Флойда запрограмований правильно.

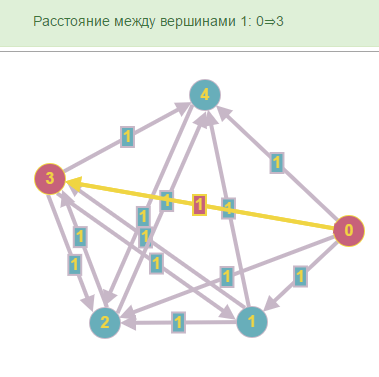


Рисунок 7.2 – Перевірка алгоритму Флойда в graphonline

б)алгоритм Данцига

Результат роботи алгоритму Данцига наведено на рисунку 7.3:

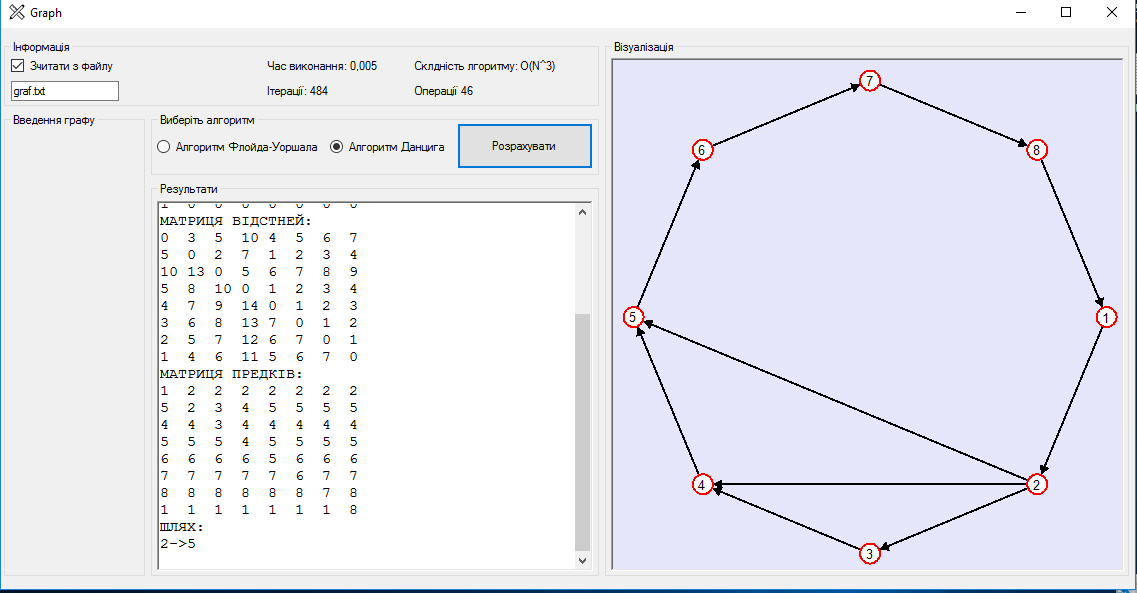


Рисунок 7.3 – Результат виконання алгоритму Данцига

Результати роботи програми відповідають результатам роботи graphonline (Рисунок 7.4), тож можна зробити висновок, що алгоритм Данцига запрограмований правильно.

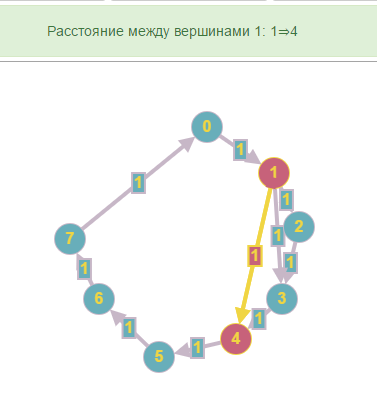


Рисунок 7.4 – Перевірка алгоритму Данцига в graphonline

## Оцінка ефективності

Для проведення тестування ефективності роботи алгоритмів Флойда і Данцига, оберемо 5 графів, на яких ми будемо тестувати наші алгоритми. Ці графи повинні мати різну кількість вершин, адже в основному швидкодія алгоритма злежить від розмірності матриці суміжності, а не її вмісту.

Для наочності результати перевірок занесені у таблиці, в яких видно, як від кількості вершин у графі залежать час роботи методів (Таблиця 7.1 – Залежність часу від розмірності (мкс)), кількість ітерацій, що відбулися (Таблиця 7.2 – Залежність кількості ітерацій від кількості вершин у графі) та кількість проведених арифметичних операцій (Таблиця 7.3 – Залежність кількості арифметичних операцій від кількості вершин у графі).

Таблиця 7.1 – Залежність часу від розмірності (мкс)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Кількість вершин графа | | | | |
| 4 | 8 | 12 | 16 | 20 |
| Флойда | 1 | 3 | 11 | 24 | 48 |
| Данцига | 0 | 3 | 10 | 23 | 46 |

Таблиця 7.2 – Залежність кількості ітерацій від розмірності

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Кількість вершин графа | | | | |
| 4 | 8 | 12 | 16 | 20 |
| Флойда | 80 | 576 | 1872 | 4352 | 8400 |
| Данцига | 58 | 484 | 1662 | 3976 | 7810 |

Таблиця 7.3 – Залежність кількості арифметичних операцій від розмірності

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Кількість вершин графа | | | | |
| 4 | 8 | 12 | 16 | 20 |
| Флойда | 7 | 28 | 108 | 144 | 365 |
| Данцига | 8 | 37 | 139 | 180 | 457 |

На основі даних таблиць побудовано графіки (Рисунок 7.1 – Залежність часу роботи алгоритмів від кількості вершин графа; Рисунок 7.2 – Залежність кількості ітерацій від кількості вершин графа; Рисунок 7.3 – Залежність кількості арифметичних операцій від кількості вершин графа).

Рисунок 7.1 – Залежність часу роботи алгоритмів від кількості вершин графа

Рисунок 7.2 – Залежність кількості ітерацій від кількості вершин графа

Рисунок 7.3 – Залежність кількості арифметичних операцій кількості вершин графа

За результатами тестування можна зробити наступні висновки:

1. Алгоритм Флойда і Данцига мють складність О(N^3) що є доволі примітивним, якщо зважати, що граф може містити тисячі вершин, проте обидва алгоритми справляються зі своєю задачею.
2. Алгоритм Данцига дещо швидший ніж алгоритм Флойда, що повязано із тим, що алгоритм Данцига заповнює матрицю відстаней поступово, на відміну від Флойда.
3. Не зважаючи на свою складність дані алгоритми досить швидко обробляють граф, адже час який потрібен алгоритмам на роботу вимірюється в (мкс).

Висновки

Для підготовки до даної курсової роботи було опрацьовано багато літератури, та інтернет ресурсів, що стосуються теорії графів, а також алгоритмів пошуку відстаней, а саме Данцига та Флойда.

Під час розробки відповідного програмного забезпечення для вирішення даної задачі, я здобув навички роботи з графами на рівні програмування, також були застосовані знання здобуті під час освоєння теоретичної частини курсової роботи, важливим також став досвід роботи з багатофайловими програмами, та графічним інтерфейсом на мові С++.

Важливою частиною написання курсової роботи також була не тільки розробка відповідних функцій, які були необхідні для роботи алгоритмів Данцига та Флойда, а й також розробка графічного інтерфейсу, який допоміг би користувачу значно легше користуватися програмою.

Наступним етапом після розробки інтерфейсу та написання функцій було тестування програми, адже часто виникають ситуації, коли користувач може помилково, або навмисно ввести не ті дані, що призводить до аварійного звершення програми, тож метою даного етапу було усунення усіх слабких місць прогрмми, що вдалося зробити поступовою обробкою кожного із можливих сценаріїв розвитку.

Кінцем праці стало готове програмне забезпечення, яке можна використовувати для розвязку задач, пов’язаних із пошуком найкоротшого маршруту, дану програму можна ткож використовувати для освітніх цілей, щоб наочно показати що таке граф, та як працюють алгоритми Флойда і Данцига.

Перелік ПОСИЛАННь

1. Алгоритм Флойда [Електронний ресурс] // e-maxx.ru – Режим доступу: <http://e-maxx.ru/algo/floyd_warshall_algorithm>
2. Орієнтований граф [Електронний ресурс] // ru.wikipedia.org – Режим доступу: <https://ru.wikipedia.org/wiki/Ориентированный_граф>
3. Метод Данцига [Електронний ресурс] // uchimatchast.ru – Режим доступу: <http://uchimatchast.ru/teory/danceg.php>
4. Работа с графами онлайн [Електронний ресурс] // graphonline.ru – Режим доступу:<http://graphonline.ru/>
5. Введение в анализ сложности алгоритмов (часть 1) [Електронний ресурс] // habrahabr.ru – Режим доступу: <https://habrahabr.ru/post/196560/>

Додаток А Технічне завдання

МІНІСТЕРСТВО НАУКИ І ОСВІТИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

автоматизованих систем обробки інформації та управління

Затвердив

Керівник Головченко М.М.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

Виконавець:

Студент Бугайов О.О.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Пошук найкоротших шляхів в мережі»

з дисципліни:

«Основи програмування»

Київ 2017

1.1 Мета: Метою курсової роботи є розробка програмного забезпечення для вирішення задачі пошуку найкоротших шляхів в мережі.

1.2 Найменування та галузь застосування об'єкта розробки:Дана робота присвячена розробці програмного забезпечення для пошуку найкоротших шляхів у мережі, а саме в графі, використовуючи алгоритми Флойда та Данцига.

1.3 Підстава для проведення робіт:Підставою для розробки програмного забезпечення є навчальний план спеціальності 6.050301 «Програмна інженерія», робоча програма дисципліни „Основи програмування”, індивідуальне завдання.

1.4 Дата початку роботи: «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2017 р.

1.5 Дата закінчення роботи: «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2017 р.

1.6 Призначення розробки: Функціональним призначенням програми є автоматизація пошуку найкоротших шляхів у мережі між вершинами графа представленого у вигляді матриці суміжності, виведення результату роботи програми на екран і в файл, візуалізація графу в наочному вигляді. Дана програма може експлуатуватися як в науковій, інженерній галузі, так і для наочного предствлення графів під час навчання.

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.

1.8 Стадії та етапи розробки:

1. Аналіз методів вирішення поставленої задачі (до \_\_.\_\_.2017 р.)

2. Розробка сценарію роботи програми (до \_\_.\_\_.2017 р.)

3. Розробка алгоритмічного забезпечення (до \_\_.\_\_.2017 р.)

4. Розробка програмного забезпечення (до \_\_.\_\_.2017 р.)

5. Розробка інтерфейсу, планування, тестування розробленої програми (до \_\_.\_\_.2017 р.)

6. Розробка пояснювальної записки (до \_\_.\_\_.2017 р.).

7. Захист курсової роботи (до \_\_.\_\_.2017 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

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення*

<Пошук найкоротших шляхів в мережі >

(Вид носія даних)

*DVD-RW*

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

*20 арк,50 Кб*

*студента групи ІП-63 І курсу*

*Бугайова О.О.*

## Лістинг програми (функціональної частини):

### Файл «MainForm.h»:

#pragma once

#define M\_PI 3.14159265358979323846

#define ITERATIONS 1000

#include <msclr\marshal\_cppstd.h>

#include <math.h>

namespace Graph {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for MainForm

/// </summary>

public ref class MainForm : public System::Windows::Forms::Form

{

public:

MainForm(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~MainForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::GroupBox^ graphPreferences;

private: System::Windows::Forms::DataGridView^ inputEdges;

protected:

protected:

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::NumericUpDown^ edgesNumber;

private: System::Windows::Forms::NumericUpDown^ vertexNumber;

private: System::ComponentModel::BackgroundWorker^ backgroundWorker1;

private: System::Windows::Forms::GroupBox^ chooseAlgorythm;

private: System::Windows::Forms::RadioButton^ rb2;

private: System::Windows::Forms::RadioButton^ rb1;

private: System::Windows::Forms::GroupBox^ resultsBox;

private: System::Windows::Forms::Button^ calculate;

private: System::Windows::Forms::RichTextBox^ results;

private: System::ComponentModel::BackgroundWorker^ backgroundWorker2;

private: System::ComponentModel::BackgroundWorker^ backgroundWorker3;

private: System::Windows::Forms::PictureBox^ picture;

private: System::Windows::Forms::GroupBox^ visualisation;

private: System::Windows::Forms::GroupBox^ inforamtion;

private: System::Windows::Forms::TextBox^ file\_name;

private: System::Windows::Forms::CheckBox^ getFile;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::NumericUpDown^ startV;

private: System::Windows::Forms::NumericUpDown^ finishV;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column2;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column3;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Column1;

private: System::Windows::Forms::Label^ label8;

private:

/// <summary>

/// Required designer variable.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(MainForm::typeid));

this->graphPreferences = (gcnew System::Windows::Forms::GroupBox());

this->label7 = (gcnew System::Windows::Forms::Label());

this->label6 = (gcnew System::Windows::Forms::Label());

this->startV = (gcnew System::Windows::Forms::NumericUpDown());

this->finishV = (gcnew System::Windows::Forms::NumericUpDown());

this->inputEdges = (gcnew System::Windows::Forms::DataGridView());

this->Column2 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column3 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Column1 = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label1 = (gcnew System::Windows::Forms::Label());

this->edgesNumber = (gcnew System::Windows::Forms::NumericUpDown());

this->vertexNumber = (gcnew System::Windows::Forms::NumericUpDown());

this->backgroundWorker1 = (gcnew System::ComponentModel::BackgroundWorker());

this->chooseAlgorythm = (gcnew System::Windows::Forms::GroupBox());

this->calculate = (gcnew System::Windows::Forms::Button());

this->rb2 = (gcnew System::Windows::Forms::RadioButton());

this->rb1 = (gcnew System::Windows::Forms::RadioButton());

this->resultsBox = (gcnew System::Windows::Forms::GroupBox());

this->results = (gcnew System::Windows::Forms::RichTextBox());

this->backgroundWorker2 = (gcnew System::ComponentModel::BackgroundWorker());

this->backgroundWorker3 = (gcnew System::ComponentModel::BackgroundWorker());

this->picture = (gcnew System::Windows::Forms::PictureBox());

this->visualisation = (gcnew System::Windows::Forms::GroupBox());

this->inforamtion = (gcnew System::Windows::Forms::GroupBox());

this->label8 = (gcnew System::Windows::Forms::Label());

this->label5 = (gcnew System::Windows::Forms::Label());

this->label4 = (gcnew System::Windows::Forms::Label());

this->label3 = (gcnew System::Windows::Forms::Label());

this->getFile = (gcnew System::Windows::Forms::CheckBox());

this->file\_name = (gcnew System::Windows::Forms::TextBox());

this->graphPreferences->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->startV))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->finishV))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->inputEdges))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->edgesNumber))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->vertexNumber))->BeginInit();

this->chooseAlgorythm->SuspendLayout();

this->resultsBox->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->picture))->BeginInit();

this->visualisation->SuspendLayout();

this->inforamtion->SuspendLayout();

this->SuspendLayout();

//

// graphPreferences

//

this->graphPreferences->Controls->Add(this->label7);

this->graphPreferences->Controls->Add(this->label6);

this->graphPreferences->Controls->Add(this->startV);

this->graphPreferences->Controls->Add(this->finishV);

this->graphPreferences->Controls->Add(this->inputEdges);

this->graphPreferences->Controls->Add(this->label2);

this->graphPreferences->Controls->Add(this->label1);

this->graphPreferences->Controls->Add(this->edgesNumber);

this->graphPreferences->Controls->Add(this->vertexNumber);

this->graphPreferences->Location = System::Drawing::Point(3, 85);

this->graphPreferences->Name = L"graphPreferences";

this->graphPreferences->Size = System::Drawing::Size(141, 464);

this->graphPreferences->TabIndex = 0;

this->graphPreferences->TabStop = false;

this->graphPreferences->Text = L"Введення графу";

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(6, 132);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(40, 13);

this->label7->TabIndex = 8;

this->label7->Text = L"Кінець";

//

// label6

//

this->label6->AutoSize = true;

this->label6->Location = System::Drawing::Point(6, 93);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(49, 13);

this->label6->TabIndex = 7;

this->label6->Text = L"Початок";

//

// startV

//

this->startV->Location = System::Drawing::Point(7, 109);

this->startV->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->startV->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->startV->Name = L"startV";

this->startV->Size = System::Drawing::Size(128, 20);

this->startV->TabIndex = 6;

this->startV->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

//

// finishV

//

this->finishV->Location = System::Drawing::Point(7, 148);

this->finishV->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->finishV->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->finishV->Name = L"finishV";

this->finishV->Size = System::Drawing::Size(128, 20);

this->finishV->TabIndex = 5;

this->finishV->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

//

// inputEdges

//

this->inputEdges->AllowUserToAddRows = false;

this->inputEdges->AllowUserToDeleteRows = false;

this->inputEdges->AllowUserToResizeColumns = false;

this->inputEdges->AllowUserToResizeRows = false;

this->inputEdges->BackgroundColor = System::Drawing::SystemColors::InactiveCaption;

this->inputEdges->ColumnHeadersHeight = 20;

this->inputEdges->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->Column2, this->Column3,

this->Column1

});

this->inputEdges->Location = System::Drawing::Point(6, 174);

this->inputEdges->Name = L"inputEdges";

this->inputEdges->RowHeadersVisible = false;

this->inputEdges->RowHeadersWidthSizeMode = System::Windows::Forms::DataGridViewRowHeadersWidthSizeMode::AutoSizeToAllHeaders;

this->inputEdges->Size = System::Drawing::Size(129, 284);

this->inputEdges->TabIndex = 4;

this->inputEdges->CellValidated += gcnew System::Windows::Forms::DataGridViewCellEventHandler(this, &MainForm::inputEdges\_CellValidated);

//

// Column2

//

this->Column2->AutoSizeMode = System::Windows::Forms::DataGridViewAutoSizeColumnMode::Fill;

this->Column2->HeaderText = L"Поч.";

this->Column2->Name = L"Column2";

this->Column2->Resizable = System::Windows::Forms::DataGridViewTriState::False;

//

// Column3

//

this->Column3->AutoSizeMode = System::Windows::Forms::DataGridViewAutoSizeColumnMode::Fill;

this->Column3->HeaderText = L"Кін.";

this->Column3->Name = L"Column3";

this->Column3->Resizable = System::Windows::Forms::DataGridViewTriState::False;

//

// Column1

//

this->Column1->AutoSizeMode = System::Windows::Forms::DataGridViewAutoSizeColumnMode::Fill;

this->Column1->HeaderText = L"Вага";

this->Column1->Name = L"Column1";

this->Column1->Resizable = System::Windows::Forms::DataGridViewTriState::False;

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(6, 55);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(87, 13);

this->label2->TabIndex = 3;

this->label2->Text = L"Кількість Ребер";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(6, 16);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(94, 13);

this->label1->TabIndex = 2;

this->label1->Text = L"Кількість вершин";

//

// edgesNumber

//

this->edgesNumber->Location = System::Drawing::Point(7, 71);

this->edgesNumber->Name = L"edgesNumber";

this->edgesNumber->Size = System::Drawing::Size(128, 20);

this->edgesNumber->TabIndex = 1;

this->edgesNumber->ValueChanged += gcnew System::EventHandler(this, &MainForm::edgesNumber\_ValueChanged);

//

// vertexNumber

//

this->vertexNumber->Location = System::Drawing::Point(7, 32);

this->vertexNumber->Maximum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 20, 0, 0, 0 });

this->vertexNumber->Minimum = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->vertexNumber->Name = L"vertexNumber";

this->vertexNumber->Size = System::Drawing::Size(128, 20);

this->vertexNumber->TabIndex = 0;

this->vertexNumber->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

this->vertexNumber->ValueChanged += gcnew System::EventHandler(this, &MainForm::vertexNumber\_ValueChanged);

//

// chooseAlgorythm

//

this->chooseAlgorythm->BackColor = System::Drawing::SystemColors::Control;

this->chooseAlgorythm->Controls->Add(this->calculate);

this->chooseAlgorythm->Controls->Add(this->rb2);

this->chooseAlgorythm->Controls->Add(this->rb1);

this->chooseAlgorythm->Location = System::Drawing::Point(150, 85);

this->chooseAlgorythm->Name = L"chooseAlgorythm";

this->chooseAlgorythm->Size = System::Drawing::Size(448, 63);

this->chooseAlgorythm->TabIndex = 1;

this->chooseAlgorythm->TabStop = false;

this->chooseAlgorythm->Text = L"Виберіть алгоритм";

//

// calculate

//

this->calculate->Location = System::Drawing::Point(306, 10);

this->calculate->Name = L"calculate";

this->calculate->Size = System::Drawing::Size(136, 46);

this->calculate->TabIndex = 2;

this->calculate->Text = L"Розрахувати";

this->calculate->UseVisualStyleBackColor = true;

this->calculate->Click += gcnew System::EventHandler(this, &MainForm::calculate\_Click);

//

// rb2

//

this->rb2->AutoSize = true;

this->rb2->Location = System::Drawing::Point(179, 25);

this->rb2->Name = L"rb2";

this->rb2->Size = System::Drawing::Size(121, 17);

this->rb2->TabIndex = 1;

this->rb2->TabStop = true;

this->rb2->Text = L"Алгоритм Данцига";

this->rb2->UseVisualStyleBackColor = true;

//

// rb1

//

this->rb1->AutoSize = true;

this->rb1->Location = System::Drawing::Point(6, 25);

this->rb1->Name = L"rb1";

this->rb1->Size = System::Drawing::Size(167, 17);

this->rb1->TabIndex = 0;

this->rb1->TabStop = true;

this->rb1->Text = L"Алгоритм Флойда-Уоршала";

this->rb1->UseVisualStyleBackColor = true;

//

// resultsBox

//

this->resultsBox->Controls->Add(this->results);

this->resultsBox->Location = System::Drawing::Point(150, 154);

this->resultsBox->Name = L"resultsBox";

this->resultsBox->Size = System::Drawing::Size(448, 395);

this->resultsBox->TabIndex = 2;

this->resultsBox->TabStop = false;

this->resultsBox->Text = L"Результати";

//

// results

//

this->results->BackColor = System::Drawing::SystemColors::ButtonHighlight;

this->results->Font = (gcnew System::Drawing::Font(L"Courier New", 11.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->results->Location = System::Drawing::Point(6, 19);

this->results->Name = L"results";

this->results->ReadOnly = true;

this->results->Size = System::Drawing::Size(436, 370);

this->results->TabIndex = 0;

this->results->Text = L"";

//

// picture

//

this->picture->BackColor = System::Drawing::SystemColors::ButtonHighlight;

this->picture->BorderStyle = System::Windows::Forms::BorderStyle::Fixed3D;

this->picture->Location = System::Drawing::Point(6, 18);

this->picture->Name = L"picture";

this->picture->Size = System::Drawing::Size(513, 513);

this->picture->TabIndex = 1;

this->picture->TabStop = false;

//

// visualisation

//

this->visualisation->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->visualisation->AutoSizeMode = System::Windows::Forms::AutoSizeMode::GrowAndShrink;

this->visualisation->BackColor = System::Drawing::SystemColors::Control;

this->visualisation->Controls->Add(this->picture);

this->visualisation->Location = System::Drawing::Point(604, 12);

this->visualisation->Name = L"visualisation";

this->visualisation->Size = System::Drawing::Size(524, 537);

this->visualisation->TabIndex = 3;

this->visualisation->TabStop = false;

this->visualisation->Text = L"Візуалізація";

//

// inforamtion

//

this->inforamtion->Controls->Add(this->label8);

this->inforamtion->Controls->Add(this->label5);

this->inforamtion->Controls->Add(this->label4);

this->inforamtion->Controls->Add(this->label3);

this->inforamtion->Controls->Add(this->getFile);

this->inforamtion->Controls->Add(this->file\_name);

this->inforamtion->Location = System::Drawing::Point(3, 12);

this->inforamtion->Name = L"inforamtion";

this->inforamtion->Size = System::Drawing::Size(595, 67);

this->inforamtion->TabIndex = 4;

this->inforamtion->TabStop = false;

this->inforamtion->Text = L"Інформація";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Location = System::Drawing::Point(408, 44);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(53, 13);

this->label8->TabIndex = 6;

this->label8->Text = L"Операції:";

//

// label5

//

this->label5->AutoSize = true;

this->label5->Location = System::Drawing::Point(408, 19);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(122, 13);

this->label5->TabIndex = 5;

this->label5->Text = L"Складність алгоритму:";

//

// label4

//

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(260, 44);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(47, 13);

this->label4->TabIndex = 4;

this->label4->Text = L"Ітерації:";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(260, 19);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(87, 13);

this->label3->TabIndex = 3;

this->label3->Text = L"Час виконання:";

//

// getFile

//

this->getFile->AutoSize = true;

this->getFile->Location = System::Drawing::Point(7, 18);

this->getFile->Name = L"getFile";

this->getFile->Size = System::Drawing::Size(109, 17);

this->getFile->TabIndex = 2;

this->getFile->Text = L"Зчитати з файлу";

this->getFile->UseVisualStyleBackColor = true;

this->getFile->CheckedChanged += gcnew System::EventHandler(this, &MainForm::getFile\_CheckedChanged);

//

// file\_name

//

this->file\_name->Location = System::Drawing::Point(7, 41);

this->file\_name->Name = L"file\_name";

this->file\_name->Size = System::Drawing::Size(108, 20);

this->file\_name->TabIndex = 1;

this->file\_name->Visible = false;

//

// MainForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::SystemColors::Control;

this->ClientSize = System::Drawing::Size(1134, 561);

this->Controls->Add(this->inforamtion);

this->Controls->Add(this->visualisation);

this->Controls->Add(this->resultsBox);

this->Controls->Add(this->chooseAlgorythm);

this->Controls->Add(this->graphPreferences);

this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));

this->MaximumSize = System::Drawing::Size(1150, 600);

this->MinimumSize = System::Drawing::Size(1150, 600);

this->Name = L"MainForm";

this->Text = L"Graph";

this->graphPreferences->ResumeLayout(false);

this->graphPreferences->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->startV))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->finishV))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->inputEdges))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->edgesNumber))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->vertexNumber))->EndInit();

this->chooseAlgorythm->ResumeLayout(false);

this->chooseAlgorythm->PerformLayout();

this->resultsBox->ResumeLayout(false);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->picture))->EndInit();

this->visualisation->ResumeLayout(false);

this->inforamtion->ResumeLayout(false);

this->inforamtion->PerformLayout();

this->ResumeLayout(false);

}

#pragma endregion

private: void printGraph(int\*\* sMatrix, int vertex) { // printing a graph

Graphics^ g = picture->CreateGraphics(); // creating graphic object in picture area

g->Clear(Color::Lavender);

Pen^ blackPen = gcnew Pen(Color::Black);

blackPen->Width = 2;

System::Drawing::Drawing2D::AdjustableArrowCap^ bigArrow = gcnew System::Drawing::Drawing2D::AdjustableArrowCap(5, 5);

blackPen->CustomEndCap = bigArrow;

Pen^ redPen = gcnew Pen(Color::Red);

redPen->Width = 2;

SolidBrush^ whiteBrush = gcnew SolidBrush(Color::White);

SolidBrush^ blackBrush = gcnew SolidBrush(Color::Black);

float step = (2 \* M\_PI) / vertex; // we are going to draw one vertex every step radians

float radius = (picture->Width - 40) / 2.0;

int i = 0;

for (float angle = 0; i < vertex; angle += step, i++) // drawing edges

{

float cs = cos(angle);

float sn = sin(angle);

float x = (radius \* cs) + picture->Width / 2.0; // getting x coord of start

float y = (radius \* sn) + picture->Height / 2.0; // getting y coord of start

for (int k = 0; k < vertex; k++)

{

if (i < vertex && sMatrix[i][k] != 0) // checking wether there is an edge beetwen start and current vertex

{

float cs2 = cos(step\*k);

float sn2 = sin(step\*k);

float x2 = ((radius \* cs2) + picture->Width / 2.0); // getting x of current vertex

float y2 = ((radius \* sn2) + picture->Height / 2.0); // getting y of current vertex

float lamda = sqrt(pow((x2 - x), 2) + pow((y2 - y), 2)) / 10;

float goalX = (x + lamda\*x2) / (1 + lamda);

float goalY = (y + lamda\*y2) / (1 + lamda);

g->DrawLine(blackPen, x, y, goalX, goalY); // drawing a line

}

}

}

i = 0;

for (float angle = 0; i < vertex; angle += step, i++) // drawing vertexes

{

float cs = cos(angle);

float sn = sin(angle);

float x = ((radius \* cs) + picture->Width / 2.0);

float y = ((radius \* sn) + picture->Height / 2.0);

g->FillEllipse(whiteBrush, x - 10, y - 10, 20.0, 20.0);

g->DrawEllipse(redPen, x - 10, y - 10, 20.0, 20.0);

System::String^ tmp;

tmp = System::Convert::ToString(i + 1);

System::Drawing::Font^ font = gcnew System::Drawing::Font("Arial", 10);

g->DrawString(tmp, font, blackBrush, x - 6, y - 7);

}

}

private: void printMatrix(int mSize, int\*\* matrix) { // printing a square matrix

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

{

for (int k = 0; k < mSize; k++)

{

if (matrix[i][k] != INFINITY\_) {

results->Text += String::Format("{0,-3}", Convert::ToString(matrix[i][k]));

}

else {

results->Text += String::Format("{0,-3}", "-");

}

}

results->Text += "\n";

}

}

private: void scanmatrix(int size, int \*\*matrix) { // scaning sMatrix if results are given in our man window

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

{

for (int k = 0; k < size; k++)

{

matrix[i][k] = 0;

}

}

for (int i = 0; i < edgesNumber->Value; i++)

{

int tmpI = Convert::ToInt32(inputEdges->Rows[i]->Cells[0]->Value) - 1;

int tmpJ = Convert::ToInt32(inputEdges->Rows[i]->Cells[1]->Value) - 1;

matrix[tmpI][tmpJ] = Convert::ToInt32(inputEdges->Rows[i]->Cells[2]->Value);

}

}

private: void showPath(int start, int finish, int\*\* fMatrix, int\*\* lMatrix) { // this function tries to get the shortest path beetwen

// and finish, if there is so

start--;

finish--;

if (lMatrix[start][finish] == INFINITY\_)

{

results->Text += "Не існує шляху між заданими вершинами\n";

return;

}

int tmp = start;

results->Text += String::Format("{0}", Convert::ToString(tmp + 1));

while (tmp != finish)

{

tmp = fMatrix[tmp][finish] - 1;

results->Text += String::Format("->{0}", Convert::ToString(tmp + 1));

}

results->Text += "\n";

}

private: bool checkGraph() { // checking graph on the case, if it is not correct

bool result = true;

// checking for right number of edges

int vertex = Convert::ToInt32(vertexNumber->Value);

int edges = Convert::ToInt32(edgesNumber->Value);

if (vertex\*(vertex - 1) < edges) {

results->Text = "Ви ввели неправильну кількість ребер;\n";

result = false;

}

//

// checking for empty edgees and right vertex

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

{

if (Convert::ToString(inputEdges->Rows[i]->Cells[0]->Value) == "" ||

Convert::ToString(inputEdges->Rows[i]->Cells[1]->Value) == "") {

results->Text = "Ви залишили незаповнені поля;\n";

result = false;

break;

}

int output = Convert::ToInt32(inputEdges->Rows[i]->Cells[0]->Value);

int input = Convert::ToInt32(inputEdges->Rows[i]->Cells[1]->Value);

if (output > vertex || input > vertex) {

results->Text = "Ви ввели неправильну вершину;\n";

result = false;

break;

}

}

//

return result;

}

private: System::Void edgesNumber\_ValueChanged(System::Object^ sender, System::EventArgs^ e) { // event handler, if user changed eges number

int vertex = Convert::ToInt32(vertexNumber->Value);

int edges = Convert::ToInt32(edgesNumber->Value);

if (vertex\*(vertex - 1) >= edges) {

inputEdges->ColumnCount = 3;

inputEdges->RowCount = Convert::ToInt32(edgesNumber->Value);

}

else {

edgesNumber->Value = vertex\*(vertex - 1);

}

}

private: System::Void calculate\_Click(System::Object^ sender, System::EventArgs^ e) { // event handler, if user pressed "calculate" button

Graphics^ g = picture->CreateGraphics(); // cleaning previous graph if it was so

g->Clear(Color::Lavender);

// variables, that represents our graph

int edges;

int mSize;

int \*\*sMatrix;

int starting, finishing;

//

if (getFile->Checked == false) //geting the type of input graph

{

if (!checkGraph()) return;

edges = Convert::ToInt32(edgesNumber->Value);

mSize = Convert::ToInt32(vertexNumber->Value);

sMatrix = new int \*[mSize];

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

sMatrix[i] = new int[mSize];

}

scanmatrix(mSize, sMatrix);

starting = Convert::ToInt32(startV->Value);

finishing = Convert::ToInt32(finishV->Value);

}

else

{

System::String^ tmp = file\_name->Text;

std::string fileName = msclr::interop::marshal\_as<std::string>(tmp);

if (!File::getGraphFromFile(sMatrix, fileName.c\_str(), mSize, edges, starting, finishing))

{

results->Text += "Помилка під час зчитування файлу\n";

return;

}

}

int \*\*lMatrix = new int \*[mSize];

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

lMatrix[i] = new int[mSize];

}

int \*\*fMatrix = new int \*[mSize];

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

fMatrix[i] = new int[mSize];

}

// our benchmark variables

clock\_t start, end;

int iterations = 0;

int operations = 0;

//

if (rb1->Checked) { // floid uorhsal

start = clock();

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

{

Floid::floid(sMatrix, mSize, lMatrix, &iterations, fMatrix, &operations);

}

end = clock();

label5->Text = "Склдність лгоритму: O(N^3)";

}

else if (rb2->Checked) { // danzig

start = clock();

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

{

Danzig::danzig(sMatrix, mSize, lMatrix, &iterations, fMatrix, &operations);

}

end = clock();

label5->Text = "Склдність лгоритму: O(N^3)";

}

else { // none checked

results->Text += "Оберіть алгоритм!\n";

return;

}

//functions, that showing the result of our program

label4->Text = "Ітерації: ";

label4->Text += Convert::ToString(iterations / ITERATIONS);

label8->Text = "Операції ";

label8->Text += Convert::ToString(operations / ITERATIONS);

label3->Text = "Час виконання: ";

label3->Text += Convert::ToString(((double)end - (double)start) / ITERATIONS / CLOCKS\_PER\_SEC \* 1000);

printGraph(sMatrix, mSize);

results->Text = "МАТРИЦЯ СУМІЖНОСТІ:\n";

printMatrix(mSize, sMatrix);

results->Text += "МАТРИЦЯ ВІДСТНЕЙ:\n";

printMatrix(mSize, lMatrix);

results->Text += "МАТРИЦЯ ПРЕДКІВ:\n";

printMatrix(mSize, fMatrix);

results->Text += "ШЛЯХ:\n";

showPath(starting, finishing, fMatrix, lMatrix);

if(!File::saveResults("results.txt", sMatrix, lMatrix, fMatrix, mSize, starting, finishing))

results->Text += "Не вдалося зберегти результати у файл\n";

//

}

private: System::Void inputEdges\_CellValidated(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

// event handler, which shows an error, if users inputs wrong data to our "input edges" area

int cellValue;

if (inputEdges->Rows[e->RowIndex]->Cells[e->ColumnIndex]->FormattedValue->ToString() == "") return;

if (!Int32::TryParse(inputEdges->Rows[e->RowIndex]->Cells[e->ColumnIndex]->FormattedValue->ToString(), cellValue) || (cellValue < 1))

{

MessageBox::Show("Ви ввели невірні дані", "Увага!",

MessageBoxButtons::OK, MessageBoxIcon::Exclamation);

inputEdges->CurrentCell->Value = "1";

inputEdges->Refresh();

}

else if (cellValue > Convert::ToInt32(vertexNumber->Value) && e->ColumnIndex != 2)

{

MessageBox::Show("Ви ввели непрвильну вершину", "Увага!",

MessageBoxButtons::OK, MessageBoxIcon::Exclamation);

inputEdges->CurrentCell->Value = Convert::ToString(vertexNumber->Value);

inputEdges->Refresh();

}

else if (e->ColumnIndex == 2 && cellValue > 50)

{

MessageBox::Show("Ви ввели непрвильну вагу", "Увага!",

MessageBoxButtons::OK, MessageBoxIcon::Exclamation);

inputEdges->CurrentCell->Value = Convert::ToString(50);

inputEdges->Refresh();

}

}

private: System::Void vertexNumber\_ValueChanged(System::Object^ sender, System::EventArgs^ e) { // event handler, which is called, when user

// change vertex number

int vertex = Convert::ToInt32(vertexNumber->Value);

int edges = Convert::ToInt32(edgesNumber->Value);

if (vertex\*(vertex - 1) >= edges) {

inputEdges->ColumnCount = 3;

inputEdges->RowCount = Convert::ToInt32(edgesNumber->Value);

}

else {

edgesNumber->Value = vertex\*(vertex - 1);

}

startV->Maximum = Convert::ToInt32(vertexNumber->Value);

finishV->Maximum = Convert::ToInt32(vertexNumber->Value);

}

private: System::Void getFile\_CheckedChanged(System::Object^ sender, System::EventArgs^ e) { // event handler, which checks whether getFile

// box is checked

if (getFile->Checked) {

file\_name->Visible = true;

label1->Visible = false;

label2->Visible = false;

vertexNumber->Visible = false;

edgesNumber->Visible = false;

inputEdges->Visible = false;

label6->Visible = false;

label7->Visible = false;

startV->Visible = false;

finishV->Visible = false;

}

else

{

file\_name->Visible = false;

label1->Visible = true;

label2->Visible = true;

vertexNumber->Visible = true;

edgesNumber->Visible = true;

inputEdges->Visible = true;

label6->Visible = true;

label7->Visible = true;

startV->Visible = true;

finishV->Visible = true;

}

}

};

}

### Файл «MainForm.cpp»:

#include <string>

#include <time.h>

#include <ctime>

#include "floid.h"

#include "danzig.h"

#include "file.h"

#include "MainForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ arg) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Graph::MainForm form;

Application::Run(%form);

}

### Файл «floid.h»:

#pragma once

#define INFINITY\_ 999999

namespace Floid {

int min(int a, int b);

void floid(int\*\* sMatrix, int mSize, int\*\* lMatrix, int\* iterations, int\*\* fMatrix, int\* operations);

}

### Файл «floid.cpp»:

#include "floid.h"

namespace Floid {

int min(int a, int b) // returning minarg

{

if (a > b) return b;

return a;

}

void floid(int\*\* sMatrix, int mSize, int\*\* lMatrix, int\* iterations, int\*\* fMatrix, int\* operations) // floid algorythm

{

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

{

for (int k = 0; k < mSize; k++)

{

if (sMatrix[i][k] == 0 && i != k)

{

lMatrix[i][k] = INFINITY\_;

fMatrix[i][k] = INFINITY\_;

}

else if (i == k)

{

lMatrix[i][k] = 0;

fMatrix[i][k] = k;

}

else

{

lMatrix[i][k] = sMatrix[i][k];

fMatrix[i][k] = k;

}

(\*iterations)++;

}

}

for (int k = 0; k < mSize; ++k)

{

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

{

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

{

if (lMatrix[i][j] > lMatrix[i][k] + lMatrix[k][j])

{

fMatrix[i][j] = fMatrix[i][k];

lMatrix[i][j] = lMatrix[i][k] + lMatrix[k][j];

(\*operations)++;

}

(\*iterations)++;

}

}

}

for (int i = 0; i < mSize; i++) // incrementation for user

{

for (int k = 0; k < mSize; k++)

{

if (fMatrix[i][k] != INFINITY\_)

{

fMatrix[i][k]++;

}

}

}

}

}

### Файл «danzig.h»:

#pragma once

#define INFINITY\_ 999999

namespace Danzig {

int min(int a, int b);

void danzig(int\*\* sMatrix, int mSize, int\*\* lMatrix, int\* iterations, int\*\* fMatrix, int\* operations);

}

### Файл «danzig.cpp»:

#include "danzig.h"

namespace Danzig {

int min(int a, int b) // return minarg

{

if (a > b) return b;

return a;

}

void danzig(int\*\* sMatrix, int mSize, int\*\* lMatrix, int\* iterations, int\*\* fMatrix, int\* operations) // danzig algorytm

{

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

{

for (int k = 0; k < mSize; k++)

{

if (sMatrix[i][k] == 0 && i != k)

{

lMatrix[i][k] = INFINITY\_;

fMatrix[i][k] = INFINITY\_;

}

else if (i == k)

{

lMatrix[i][k] = 0;

fMatrix[i][k] = k;

}

else

{

lMatrix[i][k] = sMatrix[i][k];

fMatrix[i][k] = k;

}

(\*iterations)++;

}

}

for (int m = 1; m < mSize + 1; m++)

{

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

{

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

{

if (lMatrix[i][m - 1] > lMatrix[j][m - 1] + lMatrix[i][j])

{

lMatrix[i][m - 1] = lMatrix[j][m - 1] + lMatrix[i][j];

fMatrix[i][m - 1] = fMatrix[i][j];

(\*operations)++;

}

(\*iterations)++;

}

}

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

{

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

{

if (lMatrix[m - 1][j] > lMatrix[m - 1][i] + lMatrix[i][j])

{

lMatrix[m - 1][j] = lMatrix[m - 1][i] + lMatrix[i][j];

fMatrix[m - 1][j] = fMatrix[m - 1][i];

(\*operations)++;

}

(\*iterations)++;

}

}

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

{

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

{

if (lMatrix[i][j] > lMatrix[i][m - 1] + lMatrix[m - 1][j])

{

lMatrix[i][j] = lMatrix[i][m - 1] + lMatrix[m - 1][j];

fMatrix[i][j] = fMatrix[i][m - 1];

(\*operations)++;

}

(\*iterations)++;

}

}

}

for (int i = 0; i < mSize; i++) // incrementation for user

{

for (int k = 0; k < mSize; k++)

{

if (fMatrix[i][k] != INFINITY\_)

{

fMatrix[i][k]++;

}

}

}

}

}

### Файл «file.h»:

#pragma once

#include <fstream>

#define INFINITY\_ 999999

namespace File {

bool getGraphFromFile(int \*\*&sMatrix, const char\* fileName, int& vertex, int& edges, int& starting, int& finishing);

bool saveResults(const char\* fileName, int \*\*sMatrix, int\*\* lMatrix, int\*\* fMatrix, int mSize, int start, int finish);

}

### Файл «file.cpp»:

#include "file.h"

namespace File {

bool getGraphFromFile(int \*\*&sMatrix, const char\* fileName, int& vertex, int& edges, int& starting, int& finishing)

{

std::ifstream file;

file.open(fileName);

if (!file.is\_open()) return false;

file >> vertex;

file >> edges;

file >> starting;

file >> finishing;

if (vertex\*(vertex - 1) < edges) return false;

sMatrix = new int\*[vertex];

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

{

sMatrix[i] = new int[vertex];

}

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

{

for (int k = 0; k < vertex; k++)

{

sMatrix[i][k] = 0;

}

}

int out, in, weight;

int i = 0;

while (!file.eof())

{

i++;

if (i > edges) break;

file >> out;

file >> in;

file >> weight;

sMatrix[out - 1][in - 1] = weight;

}

return true;

}

bool saveResults(const char\* fileName, int \*\*sMatrix, int\*\* lMatrix, int\*\* fMatrix, int mSize, int start, int finish)

{

std::ofstream file;

file.open(fileName);

if (!file.is\_open()) return false;

file << "МАТРИЦЯ СУМІЖНОСТІ:" << std::endl;

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

{

for (int k = 0; k < mSize; k++)

{

if (sMatrix[i][k] != INFINITY\_)

{

file.width(4);

file.setf(std::ios::left);

file << sMatrix[i][k];

}

else

{

file << "- ";

}

}

file << std::endl;

}

file << std::endl;

file << "МАТРИЦЯ ВІДСТАНЕЙ:" << std::endl;

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

{

for (int k = 0; k < mSize; k++)

{

if (lMatrix[i][k] != INFINITY\_)

{

file.width(4);

file.setf(std::ios::left);

file << lMatrix[i][k];

}

else

{

file << "- ";

}

}

file << std::endl;

}

file << std::endl;

file << "МАТРИЦЯ ПРЕДКІВ:" << std::endl;

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

{

for (int k = 0; k < mSize; k++)

{

if (fMatrix[i][k] != INFINITY\_)

{

file.width(4);

file.setf(std::ios::left);

file << fMatrix[i][k];

}

else

{

file << "- ";

}

}

file << std::endl;

}

file << std::endl;

file << "МАРШРУТ:" << std::endl;

start--;

finish--;

if (lMatrix[start][finish] == INFINITY\_)

{

file << "Не існує шляху між заданими вершинами." << std::endl;

return true;

}

int tmp = start;

file << tmp + 1;

while (tmp != finish)

{

tmp = fMatrix[tmp][finish] - 1;

file << "->" << tmp + 1;

}

return true;

}}