

Міністерство освіти і науки України
Черкаський національний університет імені Богдана Хмельницького
Факультет обчислювальної техніки, інтелектуальних та управляючих систем
Кафедра програмного забезпечення автоматизованих систем

КУРСОВА РОБОТА

з дисципліни « Програмування та алгоритмічні мови »

на тему: « Алгоритм Флойда »

Студента 1 курсу КС-181 групи
спеціальності
121 Інженерія програмного забезпечення

(код Назва)
Весельського

О.С.

(прізвище та ініціали)

Керівник: Порубльов І.М.

(прізвище та ініціали)

Оцінка за шкалою:

(національною, кількість балів, ECTS)

Члени комісії:

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

Черкаси – 2019

Зміст

Вступ.....	3
Розділ 1. Огляд способів реалізації поставленого завдання	5
1.1. Опис власне алгоритму Флойда.....	5
1.2. Огляд аналогічних продуктів.....	6
1.3. Способи введення даних.....	7
1.4. Способи виведення даних	8
Висновок до розділу 1	9
Розділ 2. Проектування основної частини програми та меню.....	10
2.1. Опис елементів HTML, необхідних для реалізації програми.....	10
2.2. Структура для зберігання елементів матриці.....	11
2.3. Опис частини, що відповідає за вирівнювання стовпчиків таблиці	12
2.4. Створення таблиці на формі.....	13
2.5. Основна частина програми.....	14
Висновок до розділу 2.....	15
Розділ 3. Реалізація програмного продукту.....	16
3.1. Інтерфейс користувача та його реалізація функціоналу елементів управління	16
3.2. Реалізація допоміжних методів.....	23
3.3. Реалізація структури для елементів матриці	25
3.4. Тестування програмного продукту.....	26
3.5. Керівництво користувача	31
Висновок до розділу 3.....	33
Висновок.....	34
Список використаної літератури	35
Додаток А – блок-схема алгоритму підрахунку кількості символів.....	1
Додаток Б – блок-схема алгоритму побудови таблиці на формі.....	2
Додаток В – блок-схема основної частини програми.....	4

Вступ

Мета:

Пройти повний шлях створення програмного продукту, дослідивши всі його етапи; вдосконалити навички пошуку методів вирішення поставленого завдання, проектування програмних продуктів, розробки власне програм, тестування, оформлення та представлення виконаної роботи; закріпити знання, засвоєні під час вивчення курсу Програмування та алгоритмічні мови та додатково потренуватися у їх практичному застосуванні.

Завдання:

Написати програму, яка отримуватиме на вхід матрицю суміжності графа у спеціальній формі, та виконуватиме над цим графом алгоритм Флойда. Процес виконання алгоритму має виводитися, а саме має виводитися поточний вигляд матриці на кожному кроці. При цьому значення, які щойно змінилися, повинні бути виділені, а при наведенні курсора мишки на будь-яке з них має з'являтися підказка, що пояснює, з яких саме існуючих шляхів було отримано той шлях, до довжини якого ця підказка відноситься.

Задачі:

1. Проаналізувати способи введення та виведення даних.
2. Проаналізувати способи реалізації спливаючих підказок.
3. Створити блок-схеми основних елементів програмного продукту.
4. Реалізувати продукт мовою програмування.
5. Протестувати продукт.
6. Написати пояснювальну записку, описавши процес створення програмного продукту та його функціонал, створити доповідь та презентацію.

Актуальність теми:

Дана програма може бути досить корисною, адже завдяки своїй наочності, покроковому виведенню та описам кожної зміни в матриці вона здатна допомогти студентам та будь-кому бажаючому розібратися в принципі роботи

алгоритму Флойда. Також вона стане в нагоді при розв'язанні будь-яких завдань, що потребують пошуку найкоротших шляхів або при перевірці правильності розв'язання таких завдань вручну.

Базові поняття теми

Граф —це пара $\langle V, E \rangle$, де $V \neq \emptyset$ —множина вершин, E —множина ребер, причому кожне ребро є парою вершин. [2, с.112]

Зважені графи передбачають, що кожному ребру приписане дійсне число, яке називають довжиною або вагою. [2, 113]

Матриця суміжності графа - матриця розміром $n \times n$, тобто і рядки, і стовпчики її відповідають вершинам. На перетині i -го рядка і j -го стовпчика ставиться 1, якщо є ребро (дуга) із вершини $v[i]$ у вершину $v[j]$, або 0, якщо такого ребра (дуги) немає. [2, с.117]

Матриця відстаней графа – аналогічно матриці суміжності, але на перетині i -го рядка і j -го стовпчика ставиться число, що позначає відстань між вершинами $v[i]$ і $v[j]$.

Відстань у графі від вершини $v[i]$ до вершини $v[j]$ (позначають $d(vi, vj)$)—це мінімальна серед довжин усіх можливих маршрутів з $v[i]$ у $v[j]$. (Іншими словами, відстань—це довжина найкоротшого маршруту.) [2, с.121]

Розділ 1. Огляд способів реалізації поставленого завдання

1.1. Опис власне алгоритму Флойда

Необхідність використання в роботі алгоритму Флойда, до того ж, використання його в якості основного алгоритму логічно випливає з самої теми роботи. Тому в першу чергу необхідно з'ясувати особливості саме алгоритму Флойда.

Алгоритм Флойда – алгоритм пошуку найкоротших шляхів у зваженому орієнтованому графі. Він розроблявся паралельно Робертом Флойдом і Стівеном Воршаллом і був майже одночасно опублікований ними обома у 1962 році. Втім, ще в 1959 році майже такий самий алгоритм був опублікований Бернардом Роєм, але це лишилось непоміченим. Даний алгоритм є досить простим в розумінні і дуже простим в реалізації мовою програмування. Ось так він виглядає на псевдокоді:

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
       $W[i][j] = \min(W[i][j], W[i][k] + W[k][j])$ 
```

На початку виконання алгоритму двовимірний масив W містить матрицю суміжності, подану в спеціальній формі: на головній діагоналі матриці стоять нулі, а для позначення відсутності ребра використовується знак нескінченності. Після виконання алгоритму цей масив міститиме матрицю відстаней даного графа. [2, 143] Тобто, знаходяться відразу усі відстані між усіма вершинами. Як видно з наведеного вище фрагменту псевдокоду, основна ідея алгоритму полягає в багаторазовому повторенні дії «перевірити, чи не буде коротше пройти з однієї вершини в іншу не по вже відомому маршруту, а пройшовши через ще одну, проміжну вершину». Якщо більше прив'язуватись до псевдокоду, то на початку роботи алгоритм враховує лише «однореберні» маршрути, на першій ітерації – ще й ті, які можуть проходити через проміжну вершину $v[0]$, на другій ітерації – ті, які можуть проходити через $v[0]$ та $v[1]$ (при чому вони можуть проходити як через обидві ці вершини, так і через будь-

яку одну з них, так і не проходити ні через яку), і так далі. В результаті з усіх можливих варіантів обираються найкоротші. [2, с.143]

Як вже згадувалося, алгоритм Флойда будує матрицю відстаней, тобто, знаходить відстані від кожної вершини до кожної. Це є і позитивною рисою, якщо якраз вся матриця відстаней і потрібна, і водночас негативною, якщо потрібні лише деякі відстані. Адже пришвидшити алгоритм відмовившись від пошуку не потрібних відстаней ніяк неможливо. [2, с.144]

Складність алгоритму завжди кубічна. Це також може бути як плюсом, так і мінусом, залежно від особливостей графа, до якого застосовується алгоритм: якщо граф щільний, то використання алгоритму Флойда зазвичай є доцільним; але якщо граф розріджений, його ефективність значно зменшується і вже краще зробити вибір на користь іншого алгоритму, наприклад, запустити з кожної вершини графа ефективну версію алгоритму Дейкстри. [2, с.144]

Втім, алгоритм Флойда має ще одну перевагу, яка може схилити до думки зробити вибір на його користь при розв'язанні певних категорій завдань: він, на відміну від того ж алгоритму Дейкстри, коректно працює з ребрами від'ємної довжини. Хоча наявність у графі циклічних маршрутів від'ємної довжини руйнує його коректність.

1.2. Огляд аналогічних продуктів

Мені вдалося знайти лише один аналог програмного продукту, розробка якого планується. [5] Завантажити саму програму мені не вдалося, зате вдалося переглянути її детальний опис. Цей продукт покроково показує застосування алгоритму Флойда. Результати виводяться безпосередньо на форму. Його функціонал дозволяє створити випадковий граф, переглянути діаграму графа, зберігати і завантажувати поточний стан програми, перезапустити програму. Які значення змінилися і чому вони змінилися показується виділенням відповідних комірок певним кольором. Втім, є і недоліки: не можна переглянути кілька станів на одному екрані, що здалося мені трохи незручним, значення довжини ребра може лежати лише в діапазоні $(-10; 10)$, а також не

надто зручно редагувати матрицю – для цього кожного разу відкривається окреме вікно. В цілому рівень складності цього продукту, на мою думку, вищий, ніж очікуваний рівень курсової роботи на першому курсі. Але з нього можна зробити висновки про те, яких проблем варто уникати, та який функціонал варто реалізувати у своїй програмі (можливість збереження даних, можливість багаторазового використання програми без закриття-відкриття).

1.3. Способи введення даних

Як вже зазначалося, програма повинна отримати на вхід матрицю суміжності графа. Першим способом, що сам запрошується, є введення з файлу. Цей спосіб, звичайно, містить значні переваги: він дозволить без жодних проблем працювати з графами з дуже великою кількістю вершин (що спричиняє велику кількість вхідних даних), також буде дуже корисним в ситуаціях, коли необхідно багаторазово запускати програму з одними і тими ж вхідними даними – в періоди між запусками програми дані спокійно зберігатимуться в файлі. Також, якщо файл з матрицею вже готовий, зчитування даних для користувача відбувається максимально просто – шляхом натискання однієї кнопки. Але файли зручні тільки при їх використанні. Перед тим же, як використовувати, їх потрібно якось створити. І тут починаються недоліки цього способу – щоб створити набір вхідних даних, користувачеві доведеться заходити в Провідник, відкривати текстовий файл якимись засобами, не пов'язаними з нашою програмою, і вельми незручним способом вводити дані в файл. Адже там не буде скільки-небудь зручної таблиці, довжини ребер доведеться вводити просто в рядок, через різні кількості знаків значення одного стовпчика не будуть знаходитися одне під одним та ще й доведеться постійно рахувати скільки чисел вже введено щоб не помилитися.

Таким чином, приходимо до висновку, що варто подумати над іншим способом введення даних, хоча і введення з файлу варто було б лишити для випадків, коли дані потрібно зберігати між різними запусками програми. Тобто, у нас вже має бути два способи введення, що передбачає якесь меню для

вибору між ними. Саме необхідність наявності меню є першим недоліком використання консолі (бо саме вона першою спадає на думку). Адже меню, виконане в консолі точно не буде зручним і зрозумілим для користувача з огляду на те, що єдиний спосіб взаємодії з ним – введення символів з клавіатури. До того ж введення з консолі хоч і вирішить проблему з необхідністю використовувати інші програми, все одно ніяк не змінить ситуацію з нерівними стовпчиками і необхідністю весь час рахувати кількість введених значень. Тому, мабуть, найкраще буде використати Windows Forms. Це дасть змогу створити зручне меню, але нам все ще потрібно визначитись з альтернативним способом введення. Просто створити текстове поле, в яке значення будуть вводитися так само як в файл чи в консоль нам не підходить, бо це теж не вирішує вищезгадані проблеми. Щоб їх вирішити, необхідно створити таблицю, в якій буде чітко визначена кількість стовпчиків і рядків, щоб виключити помилку користувача у їх підрахунку, а кожне значення вводитиметься у власне поле. На формі можна розташувати текстове поле, в яке користувач буде вводити кількість вершин графа, а потім, зчитавши цю кількість, програма будуватиме таблицю. Після цього користувач матиме змогу змінити розміри таблиці, ввівши в поле інше значення. І, відповідно, додамо кнопку для другого способу введення – з файлу.

1.4. Способи виведення даних

В даній програмі виведення даних і основна частина є нерозривно зв'язаними, адже суть програми саме в тому, щоб вивести дані з виділеними змінами і спливаючими підказками.

Для реалізації цього було прийнято рішення використати виведення у html-файл. В html спливаючі підказки реалізуються досить просто, так само нескладно реалізувати таблиці для зручної подачі матриць. Цей спосіб передбачає генерування html-коду в процесі роботи програми і записування його в окремий файл. А після завершення роботи алгоритму цей html-файл автоматично відкриється в браузері. Цей спосіб виведення, звісно, не ідеальний,

адже для перегляду даних потребує додаткову програму – браузер, який, до того ж, може довго запускатися на слабких комп'ютерах. Втім, це незначна плата за простоту і зрозумілість, яку ми одержуємо, користуючись цим способом, до того ж за використання виведення безпосередньо на форму теж можуть виникнути проблеми з продуктивністю, особливо на великих матрицях, особливо якщо виводити всі кроки на одному екрані (що може бути корисним для користувача). При використанні окремого html-файлу з цим проблем не виникне. Крім того, цей спосіб дає змогу зберігати вихідні дані скільки завгодно і переглядати їх в браузері не запускаючи нашу програму.

Висновок до розділу 1

Було виконано огляд схожих продуктів, розглянуто можливі варіанти вирішення поставленого завдання, визначено їх позитивні та негативні риси, обрано кращі та сформовано приблизний вигляд майбутнього програмного продукту, а саме:

- програма буде використовувати Windows Forms;
- буде реалізовано два способи введення даних, вручну в таблиці на формі і з файлу;
- програма буде виводити результат своєї роботи в HTML-файл, який потім відкриватиметься за допомогою браузера.

Розділ 2. Проектування основної частини програми та меню

2.1. Опис елементів HTML, необхідних для реалізації програми

В попередньому розділі ми дійшли висновку, що програма буде виконувати алгоритм Флойда, паралельно пишучи в окремий файл html-код, який потім можна буде переглянути в браузері. Щоб це реалізувати, нам знадобиться доповнювати значення відстаней, що виводяться, кодом, який відповідатиме за підказки. Щоб застосувати певний стиль лише до окремого фрагменту тексту, використовується тег `...`. У відкриваючому тегу можна дописати `title=...`, де на місці трьох крапок якраз і ввести текст, який буде на спливаючій підказці, що з'явиться при наведенні курсора на виділений span-ом фрагмент.

Також нам потрібно домогтися читабельного виведення матриць, щоб вони були рівними, а відповідні значення знаходилися одне під одним. Для цього застосуємо `<table>`. Його синтаксис наступний:

```
<table>
  <tr>
    <td>...</td>
  </tr>
</table>
```

Тут `<tr>` та `</tr>` відповідають за виділення меж рядка таблиці, а `<td>` та `</td>` - за виділення окремих елементів в рядку.

Крім того, варто звернути увагу на тег `<meta>`, який визначає метатеги, що використовуються для зберігання інформації, призначеної для браузерів і пошукових систем. Нас цікавить конкретний атрибут `charset`, що задає кодування елемента. Йому необхідно присвоїти значення `utf-8`, щоб уникнути можливих проблем з відображенням символів в різних браузерах.

2.2. Структура для зберігання елементів матриці

З попереднього підпункту очевидно, що для реалізації підказок до значень, які щойно змінилися, потрібно дописувати код (і, зрозуміло, сам текст підказки). Також потрібно писати позначки, що вказуватимуть користувачеві, що дане значення змінене. І щоб вказати компілятору для яких значень це все писати треба, а для яких не треба довелося б будувати якусь складну комбінацію операторів розгалуження, які зробили б код абсолютно нечитабельним. Щоб уникнути цього, створимо структуру для елементів матриці. В цій структурі напишемо `public override string ToString()`, в якому пропишемо виведення числа-довжини ребра разом із `string`-ом, який теж є полями структури, що зможе містити код підказок і їх текст, а на початку роботи програми містить пусте слово. А потім, в тому місці, де в звичайному алгоритмі Флойда просто знаходиться мінімальне з двох значень напишемо оператор розгалуження, при виконанні умов якого не лише змінюватиметься відстань між вершинами, а й записуватиметься `html`-код та текст у вищезгадане рядкове поле структури. Після цього оператора розгалуження буде блок виведення, який через `public override string ToString()` виведе змінені значення сумісно з частинами коду, що відповідають за підказки, а потім знову проініціалізує рядкові поля пустими словами. Завдяки цій же структурі вирішиться проблема з позначенням відсутності ребра. Оскільки у мові програмування не можна якось визначити нескінченність, доводиться шукати альтернативу у використанні максимального значення типу `int`, яке умовно приймається за нескінченність. І саме в `public override string ToString()` буде реалізовано тернарний оператор, який повертатиме число, якщо воно менше за `int.MaxValue` і знак нескінченності, якщо ні. Також можна створити булеве поле, що відповідатиме за те, чи було значення змінене на поточному кроці. І за допомогою ще двох тернарних операторів в залежності від значення цього поля по боках від числа будуть або виводитися символи «*» на позначення того, що значення змінилося, або пусті слова. Частина `html`-коду підказки, що пишеться після числа, насправді, теж можна не виносити в рядкове поле, а поставити в

залежність від вищезгаданого булевого поля, адже після числа може або писатися `` (якщо було змінене), або нічого (якщо не було змінене). А от частина коду підказки, що пишеться перед числом має бути заданою рядковим полем, бо містить текст самої підказки, який завжди буде різним.

2.3. Опис частини, що відповідає за вирівнювання стовпчиків таблиці

Щоб стовпчики таблиць в файлі-результаті мали однакову ширину, довелося написати кілька методів. Звісно, в html існують засоби, якими це можна задати ширину стовпчика, але задати її можна лише зарані відомим числом. А нам потрібно для кожної таблиці розраховувати його заново, щоб і всі значення помістилися в комірки, і таблиця була мінімальної можливої ширини. Для цього треба знайти яку ширину стовпчика вимагатиме найдовше значення в таблиці і надати таку ширину всім стовпчикам. Оскільки найдовшим буде те значення, яке складається з найбільшої кількості символів, потрібно буде створити змінну, що зберігатиме число-найбільшу кількість символів. Та щоб знайти це число, спершу потрібно познаходити довжини кожного значення ідесь їх зберегти. Додамо до структури цілочисельне поле, яке зберігатиме кількість напишемо метод, який цю кількість рахуватиме. Блок-схема цього методу наведена в додатку А. Він створюється нескладно, але нам ще потрібно звернути увагу на те, що якщо отримане методом число дорівнює `int.MaxValue`, то повертати кількість символів «1». Адже з точки зору програми `int.MaxValue` – це нескінченність, що позначається одним символом «∞». Повернемося до методу для знаходження найдовшого значення в матриці. Тут ніяких особливостей немає, просто знаходимо найбільше значення серед `num_of_symbols`-полів елементів матриці. Щодо засобів html, якими задається ширина стовпчиків, то це атрибут `width`, що пишеться всередині тегу `<col>`, що пишеться всередині `<table>`. Також в `<col>` треба вказати `span`, якому присвоїти значення кількості вершин графа. Так ми позначимо кількість стовпчиків, до яких треба застосовувати наведені характеристики (в нашому випадку лише ширина).

2.4. Створення таблиці на формі

Блок-схема алгоритму створення таблиці на формі наведена в додатку Б.

В якості таблиці використаємо елемент управління `dataGridView`. Він за промовчанням заповнює комірки `textBox`-ами, і дозволяє програмно читати значення, введені в ці `textbox`-и.

На початку роботи створюються дві змінні: `num_of_vertex` містить кількість стовпчиків та рядків необхідної нам таблиці (вводиться користувачем в текстове поле на формі), а `current_num_in_table` містить кількість стовпчиків та рядків, які на даний момент є в таблиці. Якщо таблиця будується вперше, то це буде 0, якщо не вперше, то інше значення. Потім ідуть два цикли, які прибирають непотрібні рядки та стовпчики, якщо вони є. Якщо ж таблиця будується вперше, або у поточної таблиці рядків і стовпчиків не більше, ніж у тієї, яка будується зараз, то в ці цикли програма просто не зайде. Потім оновлюється значення `current_num_in_table` (адже кількість могла змінитися), і запускається інший цикл, який вже додає стовпчики, яких не вистачає. Він працюватиме тільки якщо зараз стовпчиків менше ніж потрібно. Тобто, якщо треба прибрати зайве, спрацюють перші цикли, а умова для цього циклу вже не виконається, а якщо потрібно додати, то спрацює цей цикл, а умова для перших не виконається. Далі за умови що різниця між кількістю необхідних і наявних рядків невід'ємна додаються рядки (`dataGridView` дозволяє додати рядки не в циклі, а одним оператором).

Після цього програма забирає у користувача право редагувати елементи головної діагоналі і присвоює їм значення «0» (адже лише нулю вони і можуть дорівнювати). Потім відповідно до кількості рядків і стовпчиків встановлюються розміри елементу управління `dataGridView`, а відповідно до його розмірів – розміри форми. Потім робляться видимими кнопки для роботи з таблицею (на випадок, якщо вона створюється вперше і раніше ці кнопки були невидимими). Цей метод буде викликатися як по натисканню кнопки «Create table», так і після вибору користувачем файлу для імпортування через кнопку «Import».

2.5. Основна частина програми

Блок-схема основної частини програми наведена в додатку В.

Починає свою роботу програма зі створення двох змінних, що містять кількість вершин і кількість символів найдовшого значення в матриці значення (спочатку дорівнює одиниці, бо хоч один символ точно є). Потім запускається цикл, що перебирає всі елементи, рахує кількості їхніх символів і знаходить найбільшу з цих кількостей. Потім відкривається потік для запису в html-файл і записуються заголовки html-документа та таблиці, що міститиме початковий вигляд матриці. Наступним циклом цей поточний вигляд і виводиться.

Потім починається безпосередньо алгоритм Флойда. Стандартний вибір мінімального з двох значень в третьому циклі алгоритму замінений на оператор розгалуження. Для додавання значень використовується спеціальний метод, в якому прописаний захист одночасно від двох ситуацій – коли до умовної нескінченності (`int.MaxValue`) додається щось ще (тоді відбувається переповнення типу `int`), і коли від умовної нескінченності віднімають щось (тоді програма вважає що знайшла коротший шлях, хоча насправді ж пройти між вершинами, між яким немає шляху (а це й показує нескінченність в матриці), не можна). Завдяки методу в обох випадках, як і потрібно, повертається нескінченність.

Отже, в операторі розгалуження перевіряється, чи не є маршрут через нову проміжну вершину коротшим за вже відомий. І якщо є, то виконуються наступні дії: відстані між цими двома вершинами присвоюється нове значення (як і в стандартній реалізації), полю `changed` відповідного елемента присвоюється `true`, в поле `start_of_hint` записується код і текст підказки, перераховується кількість символів (і додається до неї 2, адже будуть ще дві «*»).

На цьому особливості третього циклу закінчуються, другий їх взагалі не має, а перший продовжує виконуватись. Знаходиться найбільша кількість символів, виводиться заголовок таблиці і в циклі починається виведення матриці. Оформлені html-кодом, що відповідає за формування таблиці,

виводяться поточні значення відстаней, а якщо поточне значення було змінене (поле `changed = true`), то полю `start_of_hint` присвоюється порожнє слово, `changed` присвоюється `false`, а кількість символів зменшується на 2 (зникли «*»).

Після завершення виконання алгоритму Флойда закривається потік виведення і відкривається файл з результатами.

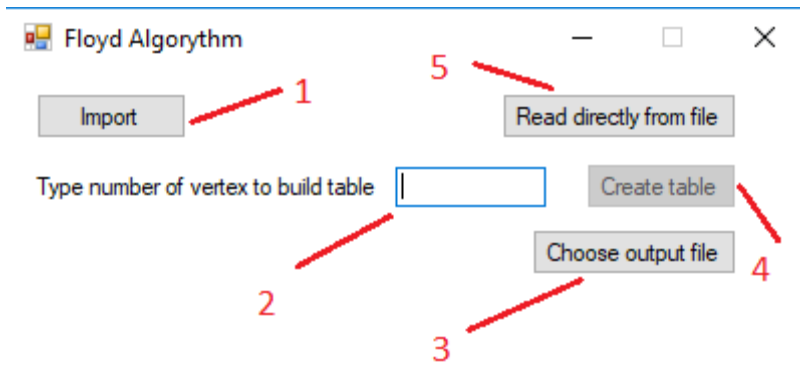
Висновок до розділу 2

Були описані основні елементи HTML, які будуть використовуватися, спроектована структура для зберігання даних. Визначено, які алгоритми будуть використовуватися в програмі, побудовані блок-схеми основних алгоритмів. Програма міститиме основний метод, що виконуватиме алгоритм Флойда і виводитиме результати, метод для побудови таблиці, два допоміжних методи, мета яких – домогтися однакової ширини стовпчиків у кінцевому файлі та метод, що запобігатиме виниканню помилок під час додавання значення до умовної нескінченності.

Розділ 3. Реалізація програмного продукту

3.1. Інтерфейс користувача та його реалізація функціоналу елементів управління

Ось так виглядає форма відразу після запуску:



1 – ця кнопка відкриває діалог для вибору файлу, з якого дані будуть імпортуватися на dataGridView.

```
private void button5_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        try
        {
            string address = openFileDialog1.FileName;
            StreamReader strrd = new StreamReader(address);
            int num_of_vertex = Convert.ToInt32(strrd.ReadLine());
            textBox2.Text = Convert.ToString(num_of_vertex);
            CreateTable();
            for (int i = 0; i < num_of_vertex; i++)
            {
                string[] temp = strrd.ReadLine().Split(' ');
                for (int j = 0; j < num_of_vertex; j++)
                {
                    dataGridView1.Rows[i].Cells[j].Value = temp[j];
                }
            }
            strrd.Close();
        }
        catch
        {

```



```

        MessageBox.Show("You have choosed incorrect file!");
    }
}

```

Тут наведений метод, який викликається по натисканню на кнопку Import. Код починає виконуватися тільки якщо користувач в діалоговому вікні натисне «ОК». В такому випадку відкривається потік для читання з файлу і відразу зчитується кількість вершин, яка в файлі знаходиться в першому рядку. Потім будується таблиця з відповідною кількості вершин кількістю стовпчиків і рядків шляхом виклику методу CreateTable(), описаного в минулому розділі. Після цього запускається цикл, який створену таблицю заповнює значеннями з файлу. Потім відбувається закриття потоку. Також реалізована обробка виключень у вигляді конструкції try catch. Оскільки тут щось може піти не так лише якщо користувач обере неправильний файл, то саме про вибір неправильного файлу програма і повідомляє.

2 – текстове поле для введення кількості вершин для створення таблиці.

```

private void textBox2_TextChanged(object sender, EventArgs e)
{
    if (textBox2.Text != "")
    {
        button4.Enabled = true;
    }
    else
    {
        button4.Enabled = false;
    }
}

```

Цей метод викликається кожного разу коли текст в TextBox-і змінюється, і перевіряє чи не виявилось після цих змін поле порожнім. Якщо виявилось, то кнопка Create table стає неактивною, якщо ж не виявилось, то активною.

3 – ця кнопка дозволяє обрати шлях для збереження файлу результату. За промовчанням збереження буде йти в папку проекту, в bin\Debug. Адреса, за якою відбувається збереження, ніяк не відображається на формі. Таке рішення було прийнято в зв'язку з тим, що навряд чи користувачеві часто знадобиться

змінювати місце збереження і взагалі якось взаємодіяти в файлом з результатом. А додаткове поле перевантажувало б форму.

```
private void button7_Click(object sender, EventArgs e)
{
    saveFileDialog2.ShowDialog();
}
```

Єдине, що робить цей метод – викликає окремий діалог, в якому користувач обирає, куди зберігати результати роботи програми. Адреса зберігається в самому saveFileDialog, звідки потім буде зчитуватись основним методом DoIt.

4 – ця кнопка натискається після введення кількості вершин щоб створити таблицю. Активна тільки коли в поле 2 що-небудь введено.

```
private void button4_MouseClick(object sender, MouseEventArgs e)
{
    CreateTable();
}
```

Все, що робить цей метод – викликає метод CreateTable.

5 – це кнопка для безпосереднього читання з файлу, без занесення даних на dataGridView. Вона призначена для ситуацій, коли граф настільки великий, що його відображення на формі може створити проблеми в роботі програми, або для ситуацій коли користувачеві не треба переглядати чи змінювати дані.

```
private void button6_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        try
        {
            StreamReader strrd = new StreamReader(openFileDialog1.FileName);
            int num_of_vertex = Convert.ToInt32(strrd.ReadLine());
            MatrixElement[,] matrix = new MatrixElement[num_of_vertex, num_of_vertex];
            for (int i = 0; i < num_of_vertex; i++)
            {
                string[] temp = strrd.ReadLine().Split(' ');
                for (int j = 0; j < num_of_vertex; j++)
                {
                    if (temp[j] == "&")
                    {
                        matrix[i, j] = new MatrixElement(int.MaxValue);
                    }
                }
            }
        }
    }
}
```

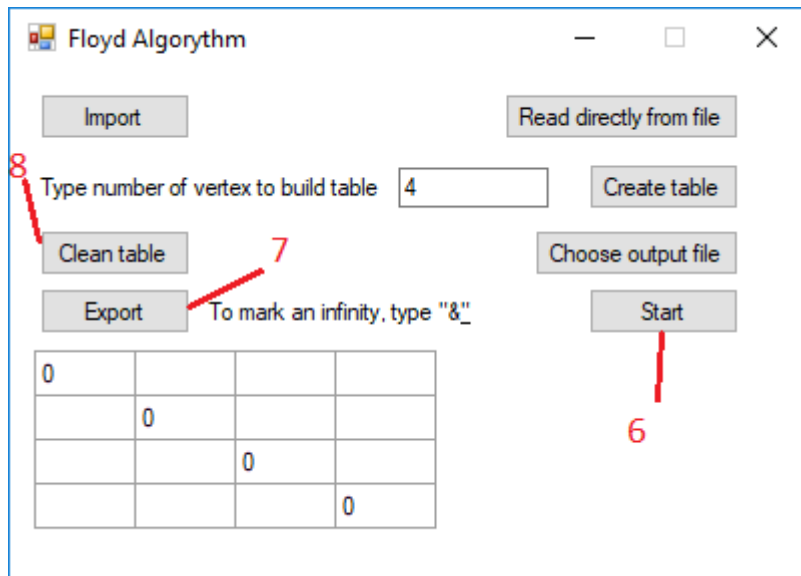
```

        }
        else
        {
            matrix[i, j] = new MatrixElement(Convert.ToInt32(temp[j]));
        }
    }
}
strrd.Close();
DoIt(matrix);
}
catch
{
    MessageBox.Show("You have choosed incorrect file!");
}
}
}

```

Цей метод відкриває openFileDialog, де користувач має обрати, з якого файлу читати вхідні дані. Після того, як буде натиснута кнопка «ОК», так само як і в Import відкривається потік для читання і зчитується кількість вершин. Далі створюється двовимірний масив з відповідною кількістю елементів, і в циклі відбувається його заповнення. Якщо прочитано «&», то в конструктор нового MatrixElement передається int.MaxValue, щоб позначити відсутність ребра, а якщо число – то передається це число. Потім закривається потік, а сформований масив передається в метод DoIt. Тут, як і в Import, реалізована обробка виключень.

Після того, як вперше будується таблиця (це може відбутися як через натискання на Create table, так і через вибір файлу в діалозі, відкритому через кнопку Import), стають видимими ще декілька кнопок для роботи з цією таблицею:



6 – ця кнопка запускає основний функціонал програми, для формування масиву використовуються дані з dataGridView.

```
private void button3_MouseClick(object sender, MouseEventArgs e)
{
    try
    {
        int num_of_vertex = dataGridView1.Columns.Count;
        MatrixElement[,] matrix = new MatrixElement[num_of_vertex, num_of_vertex];
        for (int i = 0; i < num_of_vertex; i++)
        {
            for (int j = 0; j < num_of_vertex; j++)
            {
                if (dataGridView1.Rows[i].Cells[j].Value.ToString() == "&")
                {
                    matrix[i, j] = new MatrixElement(int.MaxValue);
                }
                else
                {
                    matrix[i, j] = new
                    MatrixElement(Convert.ToInt32(dataGridView1.Rows[i].Cells[j].Value.ToString()));
                }
            }
        }
    }
}
```

```

    }
    DoIt(matrix);
    }
    catch
    {
        MessageBox.Show("Something went wrong. Maybe you entered wrong values into table or left it empty. Please enter only numbers or \"&\" and don't left cells empty.");
    }
}

```

Цей метод викликається при натисненні на кнопку Start, яка з'являється на формі тільки після створення таблиці. Метод рахує кількість стовпчиків таблиці, щоб визначити кількість вершин графа. Потім створюється двовимірний масив з такою ж кількістю рядків і стовпчиків. А далі в циклі значення з dataGridView переносяться в масив, приблизно так само, як і в Read directly from file, і так само викликається метод DoIt, в який передається сформований масив. Також тут реалізована обробка виключень у вигляді конструкції try catch.

7 – ця кнопка відкриває діалог, що дозволяє користувачу зберегти дані з таблиці

```

private void button1_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        try
        {
            string address = saveFileDialog1.FileName;
            StreamWriter strwr = new StreamWriter(address);
            int num_of_vertex = dataGridView1.Columns.Count;
            strwr.WriteLine(num_of_vertex);
            for (int i = 0; i < num_of_vertex; i++)
            {
                for (int j = 0; j < num_of_vertex; j++)
                {
                    strwr.Write(dataGridView1.Rows[i].Cells[j].Value + " ");
                }
                strwr.WriteLine();
            }
            strwr.Close();
        }
    }
}

```

```

    }
    catch
    {
        MessageBox.Show("Something went wrong. Maybe you entered wrong definitions into
        table or left it empty. Please enter only numbers or \"&\" and don't left cells
        empty.");
    }
}
}

```

Цей метод відкриває діалог, де користувач обирає шлях для експортування. Після того, як він тисне «ОК», відкривається потік для виведення, і в файл одразу записується кількість стовпчиків `dataGridView` в якості кількості вершин. Потім запускається цикл, який записує значення з таблиці в файл, розділяючи значення в рядку пробілами. Потім потік закривається. Тут також реалізована обробка виключень через `try catch`.

8 – ця кнопка стирає очищує всі комірки таблиці від значень.

```

private void button2_MouseClick(object sender, MouseEventArgs e)
{
    int num_of_vertex = dataGridView1.Columns.Count;
    for (int i = 0; i < num_of_vertex; i++)
    {
        for (int j = 0; j < num_of_vertex; j++)
        {
            dataGridView1.Rows[i].Cells[j].Value = "";
        }
        dataGridView1.Rows[i].Cells[i].Value = 0;
    }
}

```

При натисканні на кнопку `Clean table` рахується кількість стовпчиків `dataGridView`, щоб знати скільки разів повторити цикл. Сам цикл замінює всі значення в таблиці на порожні слова, а комірки на головній діагоналі заново заповнює нулями.

Також для кожної кнопки на формі реалізована підказка, яка коротко пояснює призначення кнопки. Підказка з'являється якщо трохи потримати курсор на кнопці. Код для всіх підказок однаковий за винятком самого тексту, що виводиться, тому наведемо лише один приклад.

```
private void button4_MouseHover(object sender, EventArgs e)
{
    ToolTip t = new ToolTip();
    t.SetToolTip(button4, "Create clear table on form with sizes specified");
}
```

3.2. Реалізація допоміжних методів

```
static int Plus(int a, int b)
{
    if (a == int.MaxValue || b == int.MaxValue)
    {
        return int.MaxValue;
    }
    else
    {
        return a + b;
    }
}
```

Цей метод заміняє додавання в алгоритмі Флойда. Бачимо, що в тому випадку, коли хоч один із отриманих доданків дорівнює `int.MaxValue`, повертається `int.MaxValue`. Справа в тому, що в алгоритмі Флойда нескінченність позначає відсутність шляху між вершинами. І зрозуміло, що неможливо побудувати новий маршрут, якщо, між двома вершинам цього маршруту немає ніякого зв'язку.

```
static int CountSymbols(int a)
{
    int num_of_symbols_local = 0;
    if (a < 0)
    {
        num_of_symbols_local++;
    }
    if (a == 0 || a == int.MaxValue)
    {
```

```

        num_of_symbols_local = 1;
    }
    else
    {
        while (a != 0)
        {
            a /= 10;
            num_of_symbols_local++;
        }
    }
    return num_of_symbols_local; }

```

Цей метод рахує кількість символів числа-відстані між вершинами.

Спочатку перевіряється чи не є отримане значення від’ємним. Якщо є, то потрібно збільшити лічильник кількості символів на один (щоб врахувати мінус перед числом). Якщо на вхід було отримано `int.MaxValue`, то повертається одиниця, бо `int.MaxValue` – умовна нескінченність, яка позначається одним символом. Так само повертається одиниця, якщо отримано нуль. А в загальному випадку запускається цикл який кожного разу ділить отримане число на 10 і збільшує лічильник `num_of_symbols_local`, доки число не стане нулем. Потім повертається значення лічильника.

```

static int FindMax(MatrixElement[,]matrix)
{
    int max_num_of_symbols = 1;
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            if (matrix[i, j].num_of_symbols > max_num_of_symbols)
            {
                max_num_of_symbols = matrix[i, j].num_of_symbols;
            }
        }
    }
    return max_num_of_symbols;
}

```

Цей метод шукає, в якого елемента матриці найбільша кількість символів для виведення. Спочатку створюється змінна `max_num_of_symbols`, яка і

міститиме цю кількість. Спочатку вона ініціалізується одиницею. Потім запускається цикл, який перебирає num_of_symbols-поля всіх елементів матриці, і якщо якесь з них більше за max_num_of_symbols, то це значення йому присвоюється. Потім max_num_of_symbols повертається.

3.3. Реалізація структури для елементів матриці

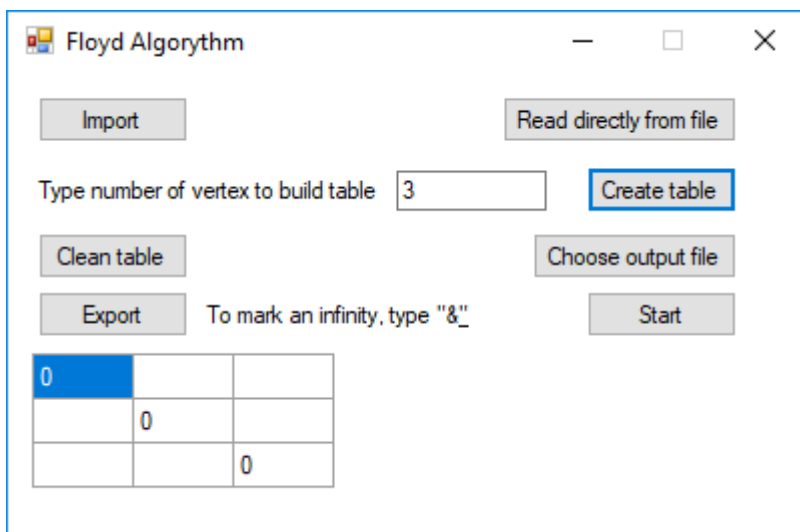
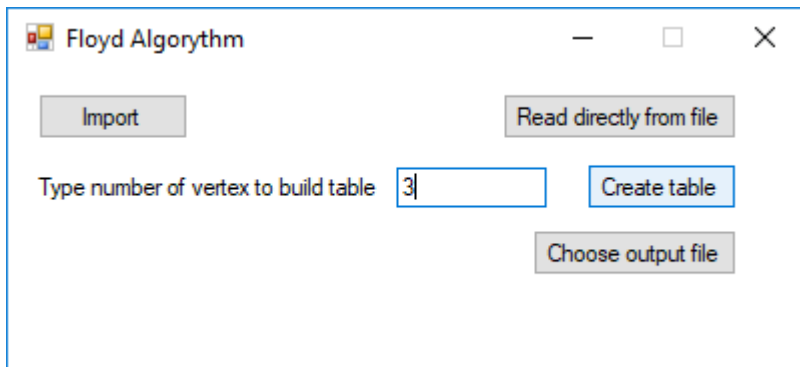
```
struct MatrixElement
{
    public int edge_weight, num_of_symbols;
    public string start_of_hint;
    public bool changed;
    public MatrixElement(int edge_weight_)
    {
        edge_weight = edge_weight_;
        changed = false;
        start_of_hint = "";
        num_of_symbols = 1;
    }
    public override string ToString()
    {
        return "<td>" + start_of_hint + (changed ? ("*") : "") +
            (edge_weight < int.MaxValue ? ("" + edge_weight) : "∞") +
            (changed ? ("*") : "") + (changed ? ("</strong></span>") : "") + "</td>";
    }
}
```

Ця структура містить два цілочисельних поля: одне (edge_weight) відповідає безпосередньо за число-відстань між вершинами, а інше (num_of_symbols) – за кількість символів, що виведуться. Також одне рядкове поле (start_of_hint), що містить код і текст підказки, і одне булеве (changed), що позначає, чи був елемент змінений на поточному кроці алгоритму Флойда. Конструктор структури отримує на вході одне число-відстань, ініціалізує num_of_symbols одиницею, start_of_hint – порожнім словом, а changed – false. В public override string ToString() повертається html-код, що позначає початок окремої комірки поточного рядка таблиці, код підказки з її текстом, зірочка, якщо значення було змінено, числове значення, якщо воно не дорівнює int.MaxValue (якщо

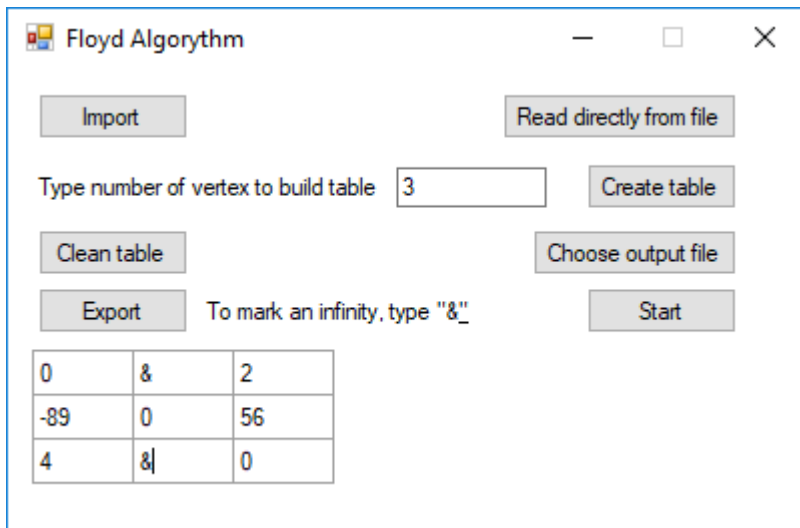
дорівнює, то « ∞ »), ще одна зірочка (знову ж, лише якщо значення змінювалося), код закінчення напівжирного тексту і ділянки, до якої застосовується підказка (якщо значення змінювалося) і код закінчення комірки таблиці.

3.4. Тестування програмного продукту

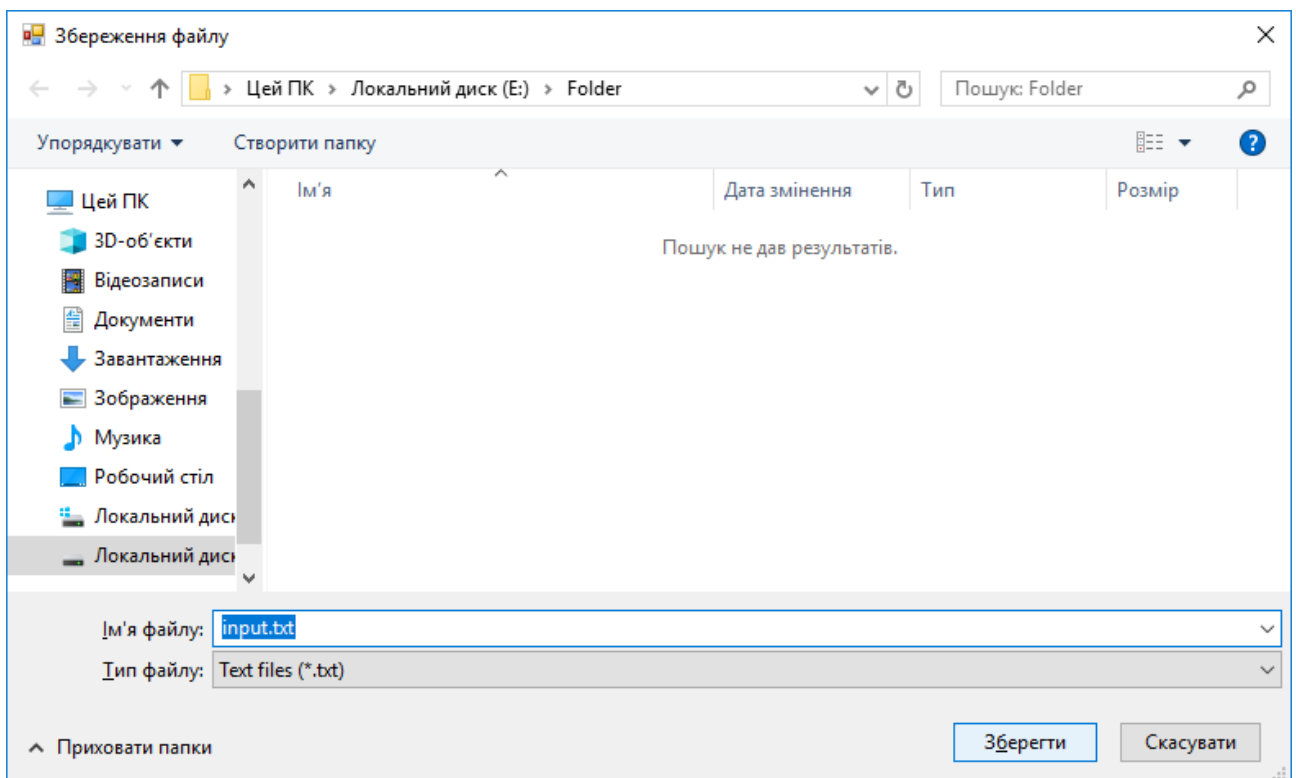
Щоб ефективно протестувати програмний продукт, проробимо наступні дії:
-створимо таблицю



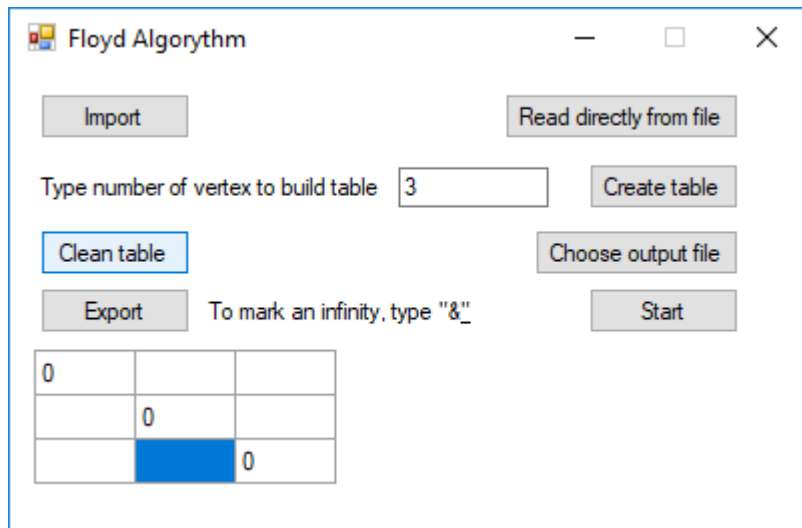
-заповнимо її



-збережемо в файлі



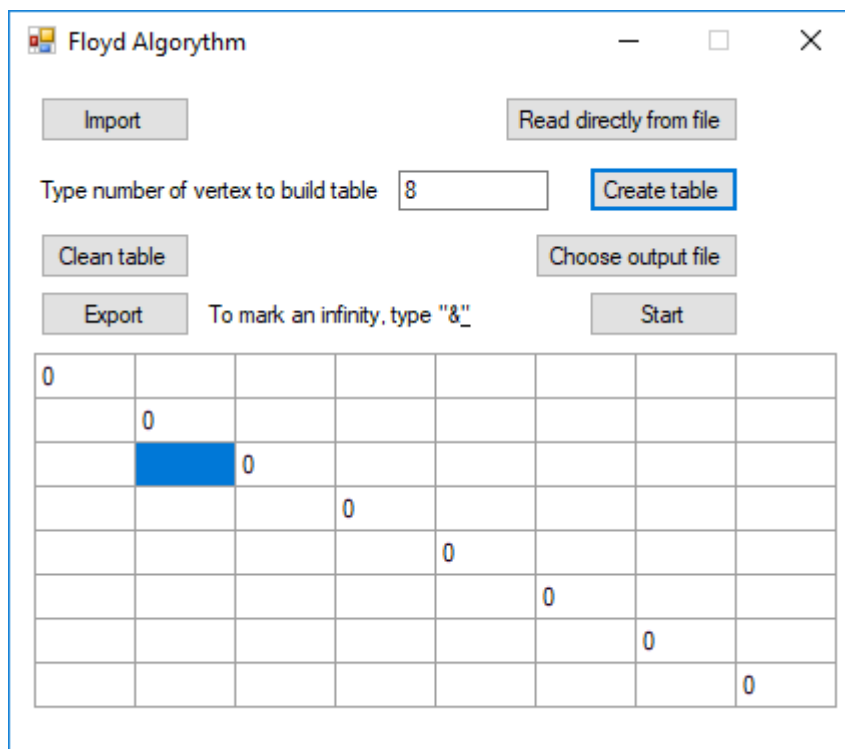
-очистимо таблицю



The screenshot shows the 'Floyd Algorithm' window. It has a title bar with a standard Windows icon, a minus sign, a maximize button, and a close button. The interface includes several buttons: 'Import', 'Read directly from file', 'Clean table' (highlighted with a blue border), 'Export', 'To mark an infinity, type "&_"', 'Choose output file', 'Start', and 'Create table'. A text input field labeled 'Type number of vertex to build table' contains the number '3'. Below the buttons is a 3x3 table with the following values:

0		
	0	
		0

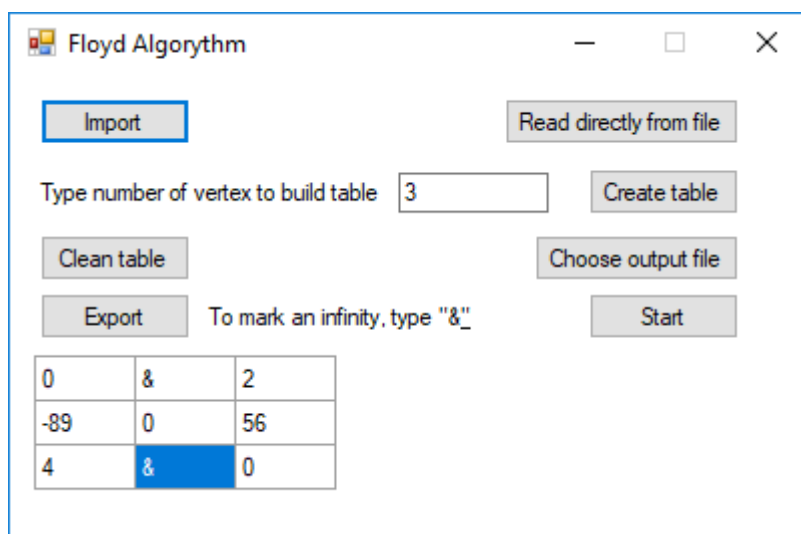
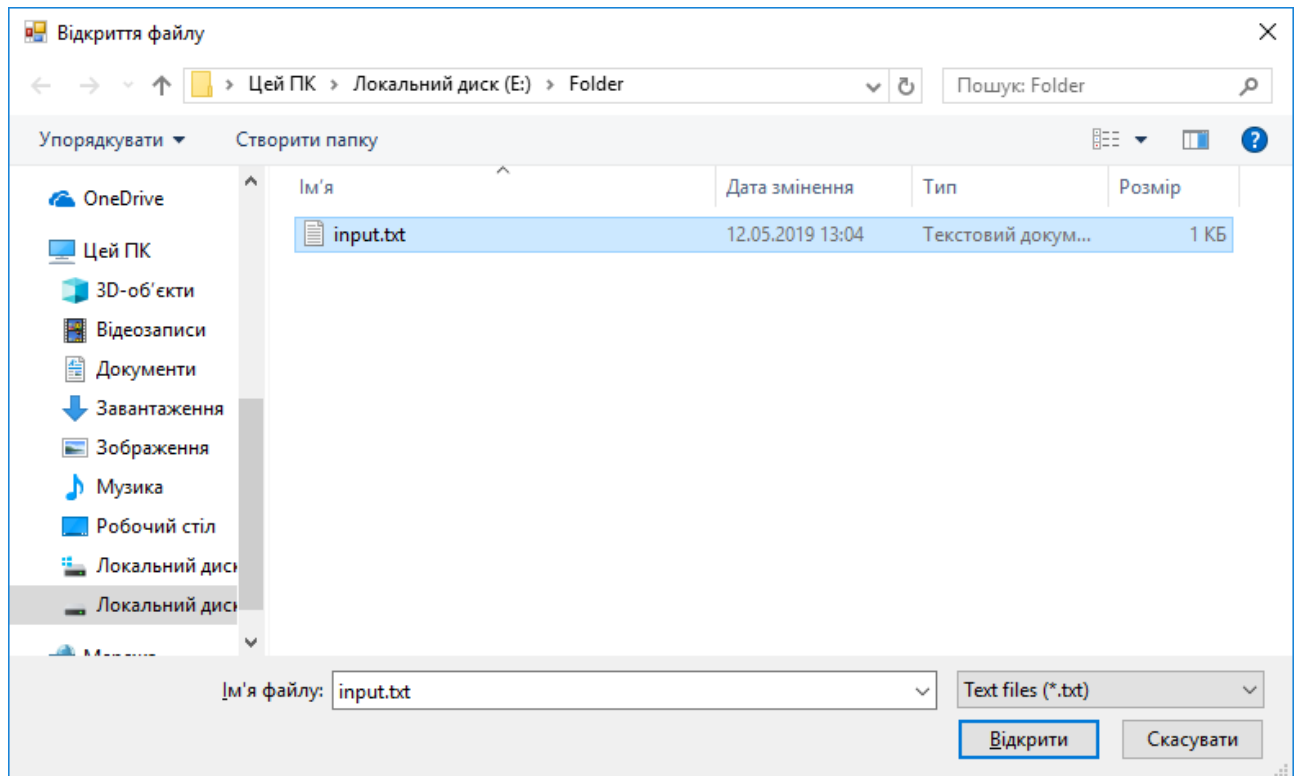
-змінимо її розміри



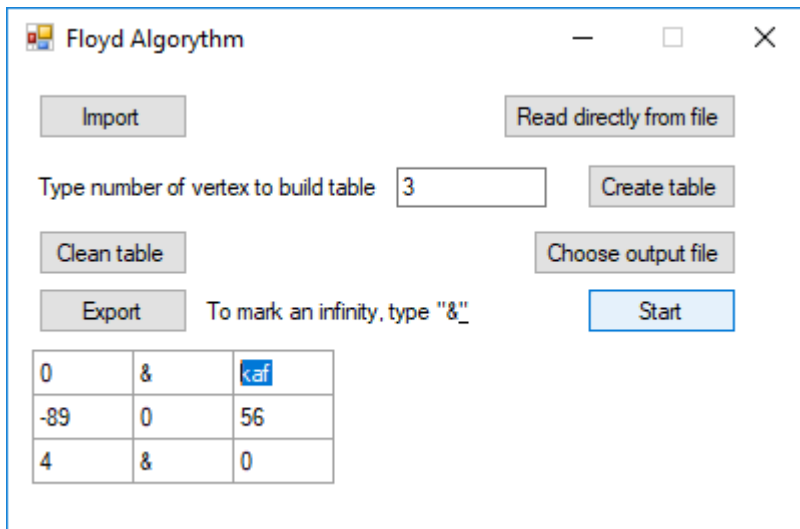
The screenshot shows the 'Floyd Algorithm' window with the same interface as the previous one. The 'Type number of vertex to build table' input field now contains the number '8'. The 'Clean table' button is still highlighted. The 8x8 table below has the following values:

0							
	0						
		0					
			0				
				0			
					0		
						0	
							0

-імпортуємо раніше збережені дані

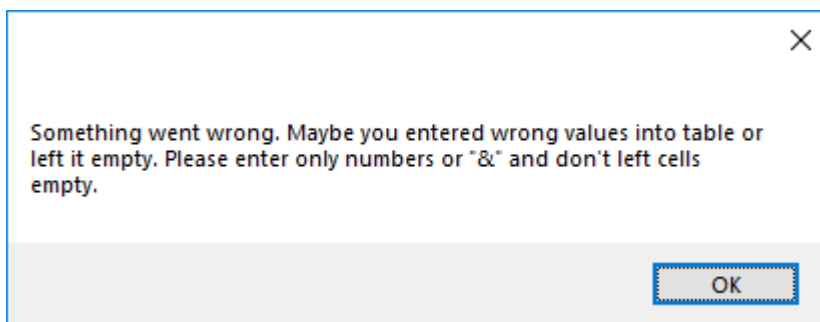


-спробуємо спровокувати аварійне завершення, змінивши значення на неправильне і запустивши алгоритм



The screenshot shows the 'Floyd Algorithm' window. The 'Type number of vertex to build table' field contains the value '3'. The table below contains the following data:

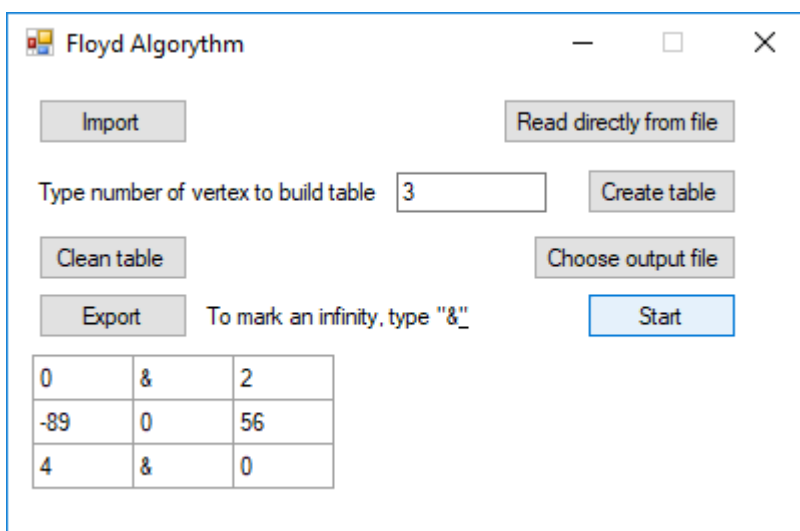
0	&	kaf
-89	0	56
4	&	0



Something went wrong. Maybe you entered wrong values into table or left it empty. Please enter only numbers or "&" and don't left cells empty.

OK

-виправимо некоректне значення, запустимо алгоритм і подивимось на результат



The screenshot shows the 'Floyd Algorithm' window with the same controls as before. The table below now contains the following data:

0	&	2
-89	0	56
4	&	0

Початковий вигляд матриці, підготовленої для алгоритма Флойда:

0	∞	2
-89	0	56
4	∞	0

Вигляд матриці наприкінці 1-ої ітерації алгоритму Флойда:

0	∞	2
-89	0	*-87*
4	∞	0

$v[1] \rightarrow (-87) \rightarrow v[2]$ отримано з $v[1] \rightarrow (-89) \rightarrow v[0] \rightarrow (2) \rightarrow v[2]$

Вигляд матриці наприкінці 2-ої ітерації алгоритму Флойда:

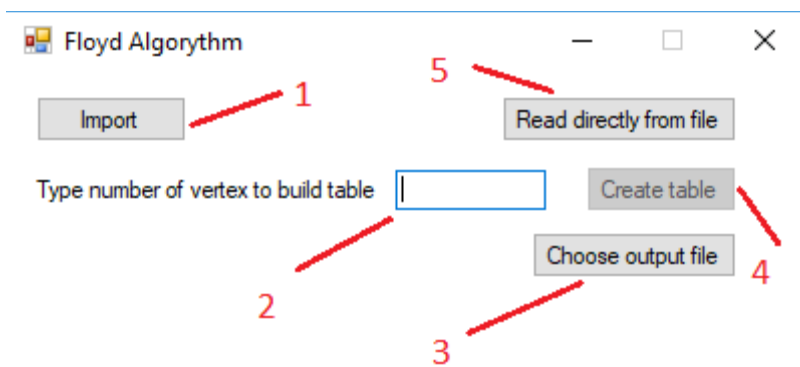
0	∞	2
-89	0	-87
4	∞	0

Вигляд матриці наприкінці 3-ої ітерації алгоритму Флойда:

0	∞	2
-89	0	-87
4	∞	0

3.5. Керівництво користувача

Після запуску програма виглядає наступним чином:



В даному стані для користувача доступні наступні дії:

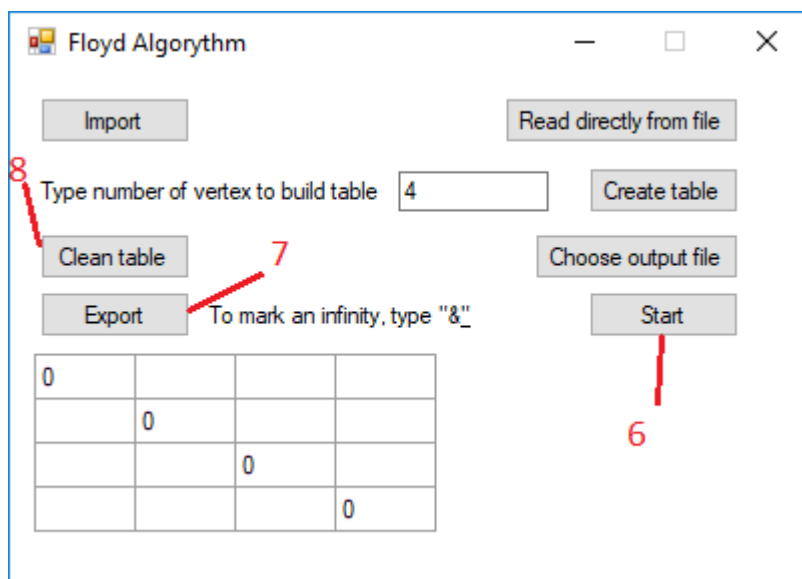
1. Кнопка Import. При натисканні на цю кнопку відкривається вікно, в якому можна вибрати файл з вхідними даними. Після натискання «ОК» на вікні програми з'явиться таблиця, в якій знаходитимуться дані з файлу з можливістю редагування.
2. В це поле можна ввести кількість вершин графа, матрицю якого користувач збирається будувати. Після введення числа стане активною кнопка 4 (її функціонал наведений нижче).

3. Ця кнопка відкриває вікно, в якому можна обрати, куди зберігатиметься вихідний файл. Цей файл відкривається автоматично, тому, якщо користувач не має наміру якось використовувати його після завершення робочої сесії в програмі, використовувати її немає сенсу.

4. Ця кнопка стає активною тільки коли в поле 2 щось введено. Вона дозволяє побудувати нову таблицю для введення даних, або змінити розміри існуючої. Кількість рядків і стовпчиків таблиці дорівнюватиме числу, введеному в поле 2.

5. Ця кнопка корисна якщо користувач хоче, щоб для алгоритму були використані дані з файлу, але йому не потрібно виводити їх на таблицю у вікні (як робить кнопка 1). Вона теж відкриє вікно для вибору файлу, а відразу після натискання «ОК» запустить основну частину програми.

Після того, як була вперше створена таблиця (неважливо, чи шляхом імпортування даних з файлу через кнопку 1, чи шляхом створення порожньої таблиці через кнопку 4), з'являється ще кілька кнопок.



6. Натискання цієї кнопки запустить основну частину програми, при чому в якості вхідних даних буде використано те, що зараз введено в таблицю в вікні.

7. Ця кнопка дозволяє зберегти те, що зараз введено в таблицю в вікні, в текстовий файл, який згодом можна буде завантажити за допомогою кнопки 1 або 5. При натисканні відкриється вікно, де потрібно буде обрати, в якій папці зберегти файл.

8. Ця кнопка дозволяє очистити таблицю від значень, тобто, робить комірки порожніми.

Щоб отримати інформацію про призначення кнопки всередині самої програми, можна навести курсор миші на кнопку і побачити підказку.

Після того, як програма завершує виконання своєї основної частини, файл з результатом буде автоматично відкритий в браузері.

В разі введення неправильних даних в таблицю чи в поле для кількості вершин, або вибору неправильного файлу з вхідними даними, програма покаже вікно з повідомленням про помилку. В такому разі потрібно натиснути «ОК» і виправити дані, що вводяться.

Висновок до розділу 3

Була написана основна частина програми, декілька допоміжних методів, створений інтерфейс користувача, прописаний функціонал всіх елементів управління. Програма дозволяє будувати таблиці для введення вхідних матриць прямо на формі, зберігати введені матриці в файл, виводити з файлу. Створена структура для зберігання даних і доданий захист від аварійного завершення і повідомлення для користувача з описом проблеми, що виникла. Написана програма протестована, помилок в роботі не було виявлено.

Висновок

Отже, програмний продукт був розроблений відповідно до поставленого завдання. Програма, як і вимагалось, покроково виводить процес виконання алгоритму Флойда, виділяє значення, які щойно змінилися, а при наведенні курсора на ці значення показує спливаючу підказку, яка описує, чому це значення змінилося і чому саме на таке. Це все реалізовано за допомогою того, що програма пише html-код в окремий файл, який потім відкривається браузером.

Взаємодія з самою програмою відбувається через інтерфейс користувача. Реалізовано можливість створювати таблицю для введення даних прямо на формі. Також дані в таблицю можна імпортувати з файлу, або дані з таблиці експортувати в файл. Таблицю можна автоматично очищувати, можна обирати шлях для збереження файлу з результатом. Також є можливість зчитати дані безпосередньо з файлу, без виведення на таблицю на формі. Вікно програми автоматично масштабується відповідно до розмірів таблиці. При введенні некоректних даних або спробі використати некоректний вхідний файл програма не завершує роботу і інформує користувача про помилку.

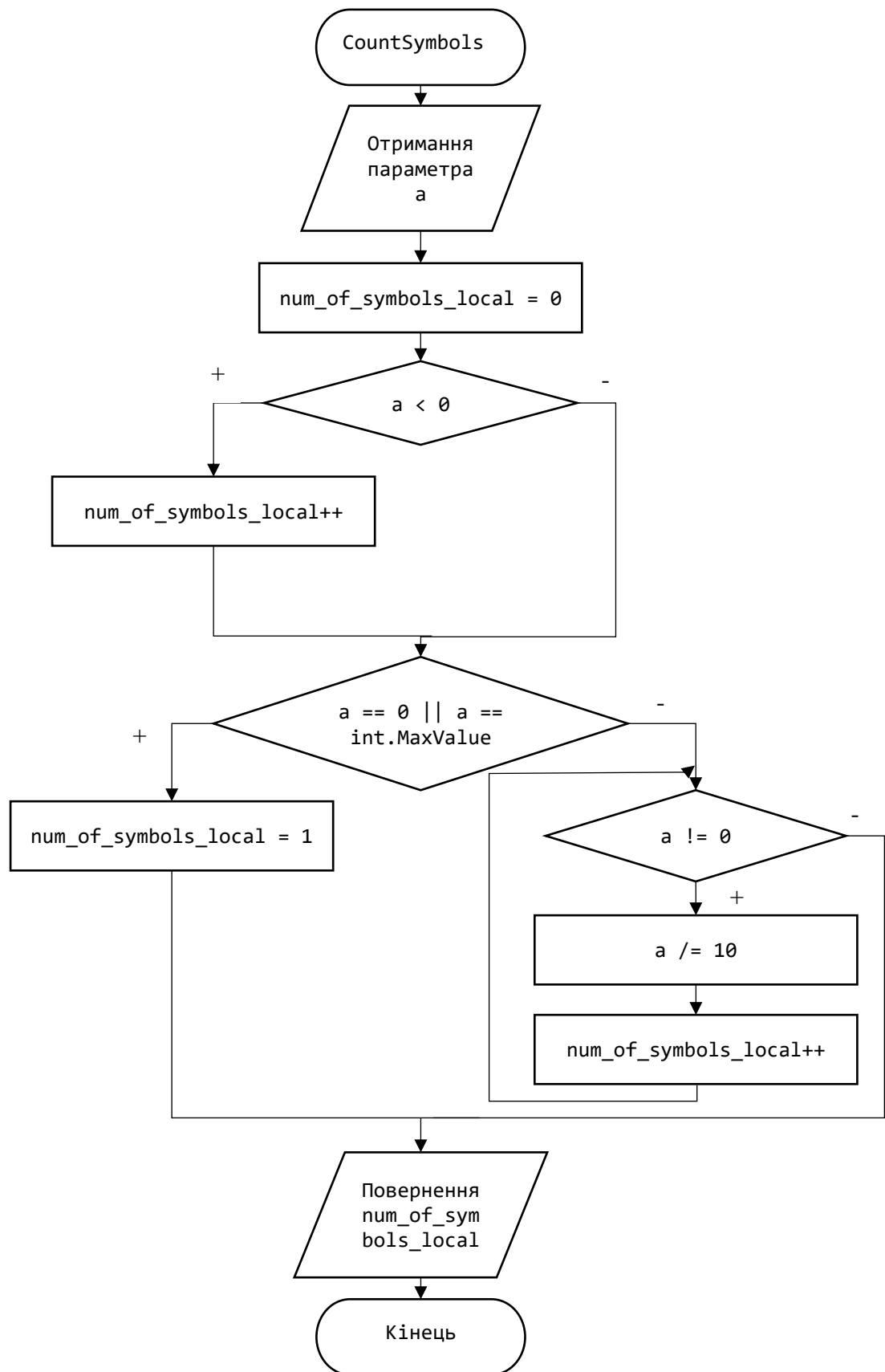
Створений програмний продукт може бути корисним в першу чергу для освітньої мети, адже можливість на прикладі розглянути роботу алгоритму Флойда, та побачити, чому і як змінюються значення, допоможе легко зрозуміти його суть. Також за допомогою цієї програми можна розв'язувати будь-які задачі, що потребують пошуку найкоротших шляхів, або, розв'язавши таку задачу вручну, перевіряти правильність розв'язку.

Список використаної літератури

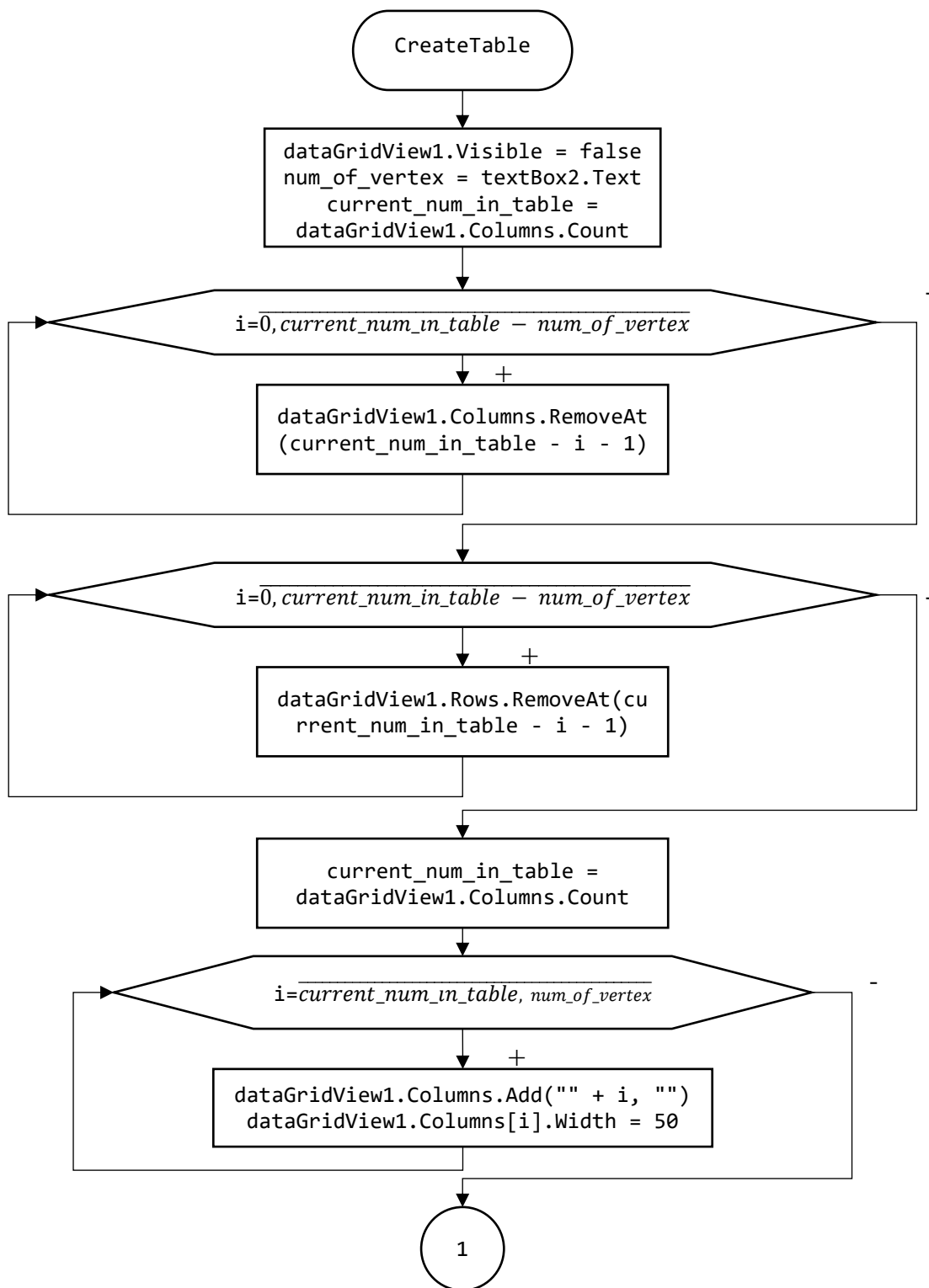
1. Гребенович Ю.Є., Онищенко Б.О., Супруненко О.О. Методичні вказівки до виконання та оформлення курсової роботи з дисциплін «Основи програмування», «Програмування та алгоритмічні мови», «Алгоритмізація та програмування» для студентів, які навчаються за напрямками підготовки 050101 – «Комп'ютерні науки», 050103 – «Програмна інженерія» та 040303 – «Системний аналіз» усіх форм навчання. [Текст] – Черкаси: Вид. від. ЧНУ імені Богдана Хмельницького, 2015. – 32 с.
2. Порубльов І.М. Дискретна математика. Навчальний посібник для студентів 1-го курсу бакалаврату галузі знань «Інформаційні технології» та споріднених. [Текст] – Черкаси: , 2018. – 220 с.
3. Алгоритм Флойда — Уоршелла [Електронний документ]. Режим доступу: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A4%D0%BB%D0%BE%D0%B9%D0%B4%D0%B0_%E2%80%94%D0%A3%D0%BE%D1%80%D1%88%D0%B5%D0%BB%D0%BB%D0%B0. Перевірено: 12.05.2019.
4. Алгоритм Флойда — Уоршелла [Електронний документ]. Режим доступу: <https://habr.com/ru/post/105825/>. Перевірено: 12.05.2019.
5. Илья Колыхматов. Алгоритм Флойда-Уоршалла. Описание интерфейса. [Електронний документ]. Режим доступу: <http://rain.ifmo.ru/cat/view.php/vis/graph-paths/floyd-warshall-2004/interface>. Перевірено: 12.05.2019.
6. Справочник по HTML. [Електронний документ]. Режим доступу: <http://htmlbook.ru/html>. Перевірено: 12.05.2019.
7. Элемент управления dataGridView. [Електронний документ]. Режим доступу: https://www.bestprog.net/ru/2018/02/17/the-datagridview-control_ru/. Перевірено: 12.05.2019.

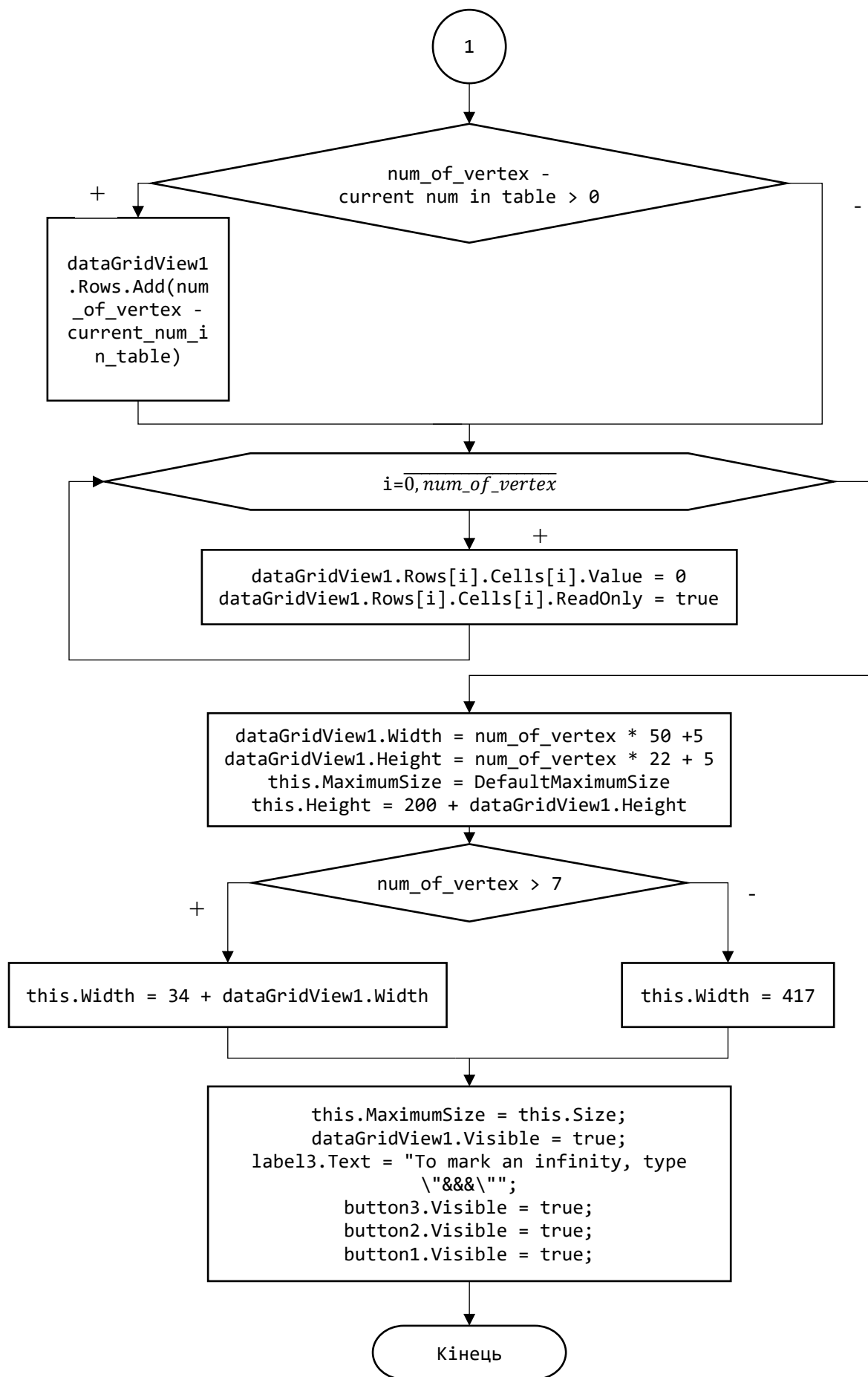
8. Как извлечь данные из определенной ячейки dataGridView. [Электронный документ]. Режим доступа: <http://www.cyberforum.ru/windows-forms/thread559983.html>. Проверено: 12.05.2019.

Додаток А – блок-схема алгоритму підрахунку кількості символів



Додаток Б – блок-схема алгоритму побудови таблиці на формі





Додаток В – блок-схема основної частини програми

